2680

Bioinformática: Búsqueda de genes en el servidor local usando información Biomédica de Bases de Datos Remotas.

Memoria del Proyecto
de Ingeniería Informática
realizado por
David Expósito Pérez
y dirigido por
Jordi Gonzàlez i Sabaté
y codirigido por
Mario Huerta

Bellaterra, 22 de Junio de 2011

El sotasignat, Jordi Gonzàlez i Sabaté

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en David Expósito Pérez

I per tal que consti firma la present.

Signat:

Bellaterra, 22 de Junio de 2011

El sotasignat, Mario Huerta

de l'empresa, Institut de Biotecnologia i de Biomedicina de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat l'empresa sota la seva supervisió mitjançant conveni amb la Universitat Autònoma de Barcelona.

Així mateix, l'empresa en té coneixement i dóna el vist-i-plau al contingut que es detalla en aquesta memòria.

Signat:

Bellaterra, 22 de Junio de 2011

ÍNDICE DE CONTENIDO

1. Introducción

1.1. Motivación personal

Dentro del campo de la bioinformática, es muy necesario encontrar aplicaciones sobre el estudio de la información génica, motivo por el cual el presente proyecto ha despertado mi interés. El proyecto que nos abarca ofrece a los usuarios herramientas para acceder a bases de datos biomédicas con la finalidad de facilitar el estudio de las relaciones existentes entre genes. Estas relaciones se añadirán a las relaciones de expresión génica que está analizando el usuario. Esta apertura, estableciendo nuevas conexiones entre los genes puede abrir nuevas vías de investigación en el estudio de patologías que hoy en día azotan a nuestra sociedad.

Es por ello que en un primer momento estaba fascinado en realizar este proyecto, porque estamos ampliando las funcionalidades de unas herramientas muy interesantes con el objetivo de potenciar las posibilidades que ofrecían en ese momento.

El centro de investigaciones del IBB, Instituto de Biotecnología y Biomedicina, es pionero en el desarrollo de estas herramientas. Existen centros de investigación que poseen bases de datos con gran cantidad de información biomédica, como publicaciones médicas en biología, implicación de los genes en las diferentes enfermedades humanas, interacciones entre proteínas, etc. Es por ello necesario desarrollar aplicativos, como los desarrollados en el IBB, para poder utilizar de forma inteligente y automática toda esta información.

Al margen de lo comentado anteriormente, este proyecto requería aplicar muchos de los conocimientos que he ido adquiriendo en la carrera, como pueden ser AJAX, Java, php, MySQL, etc.

Una de las fases importantes del proyecto, es la estructuración y el mantenimiento de las bases de datos locales que permiten usar la información de las bases biomédicas remotas. Una buena estructuración de las bases de datos locales implica reducir el coste de tiempo de acceso a la información. Debemos de tener en cuenta el gran volumen de información recopilada que se ha de organizar. Por este motivo, he considerado la aplicación a desarrollar como un reto personal ya que es un ámbito en el mundo de la Ingeniería Informática sobre el que quiero ampliar mis conocimientos de cara a mi vida profesional.

En definitiva, con este proyecto, he podido aplicar los conocimientos adquiridos durante la carrera y, de este modo, ganar experiencia. Además, la dificultad que suponía afrontar un trabajo externo a la facultad, ha resultado muy gratificante.

1.2. Estado del arte

El problema de determinar la función de una secuencia de DNA puede ser abordado desde diferentes vertientes, análisis de homología, búsqueda de motivos secuenciales, técnicas experimentales, etc.

Otra aproximación al problema puede ser la extracción de conocimiento a partir del análisis de los datos de *microarrays*. La tecnología de *microarrays* proporciona los niveles de expresión de los genes para diferentes condiciones experimentales. Al montar una *microarray* se nos proporcionará una matriz donde las filas representan genes, las columnas son experimentos y las celdas, la expresión de dicho gen para dicho experimento.

Para encontrar patrones de relación entre los genes de una *microarray* se utilizan métodos de **Data Mining** sobre los datos generados. Desde este ángulo, el **Data Mining** aplica una dinámica que se mueve en sentido contrario al método científico tradicional que se basa en formular hipótesis, por parte de los investigadores, y realiza los experimentos que confirmen o refuten la hipótesis planteada. Este es un proceso, que realizado de forma rigurosa, debe generar nuevos conocimientos. En el **Data Mining**, por el contrario, se captan y procesan los datos con la esperanza de que surja una hipótesis apropiada. Se desea que los datos nos describan o indiquen el por qué presentan determinada configuración y comportamiento.

El IBB [1], centro de investigación de la Universidad Autónoma de Barcelona (UAB), tiene una linea de investigación para el análisis de *microarrays* por **Data Mining** [2][3][4][5][6][8]. En relación a esta linea de investigación disponemos del servidor en el IBB http://revolutionresearch.uab.es [7]. En este servidor podemos encontrar la aplicación web *PCOPGene*[2], que permite estudiar y analizar *microarrays* con diversos métodos desarrollados en el IBB.

Entre las herramientas web que nos ofrece PCOPGene, encontramos una interfaz gráfica con un grafo interactivo que nos muestra la relación entre todos los genes de la *microarray* del usuario mediante una *gene network* [2]. Esta aplicación facilita el análisis de las relaciones entre los genes para la *microarray* estudiada.

Esta aplicación web, PCOPGene[2], también permite el cruce de los resultados de analizar una *microarray* dada con bases de datos biomédicas remotas. Esta funcionalidad resulta muy interesante para los investigadores, pues permite encontrar relación entre genes de la *microarray* más allá de las relaciones de expresión que nos proporciona la *microarray* analizada.

Estas relaciones entre genes (o **relaciones génicas**) pueden ser obtenidas de diferentes bases de datos:

- Bases de datos que almacenan información sobre las interacciones de las proteínas.
- Bases de datos con las publicaciones de estudios biomédicos, donde se muestran y discuten el resultado de multitud de trabajos experimentales.
- Bases de datos sobre vías de regulación de genes, es decir, genes cuyas proteínas activan a otros genes para que pasen a expresarse.
- Bases de datos con los genes clasificados por su presunta función, presumible ubicación en la célula, etc.

Estas bases de datos son facilitadas por diferentes centros de investigación vía Internet. Unos de los más importantes son *National Center for Biotechnology Information* (NCBI)[9] en Estados Unidos y KEGG (Kyoto Encyclopedia of Genes and Genomes)[10] en Japón.

El Centro Nacional para la Información Biotecnológica o National Center for Biotechnology Information (NCBI)[9] es parte de la Biblioteca Nacional de Medicina de Estados Unidos, una rama de los Institutos Nacionales de Salud. Está localizado en Bethesda, Maryland y fue fundado el 4 de noviembre de 1988 con la misión de ser una importante fuente de información de biología molecular. Almacena y constantemente actualiza la información referente a secuencias genómicas en GenBank, un índice de artículos científicos referentes a biomedicina, biotecnología, bioquímica, genética y genómica en PubMed, una recopilación de enfermedades humanas y los genes implicados en OMIM, una clasificación de los genes por función y localización de dicha función dentro de la célula en GO, además de otros datos biotecnológicos de relevancia en diversas bases de datos.

Todas las bases de datos del NCBI [9] están disponibles en línea de manera gratuita y pueden ser consultadas usando el buscador *Entrez* [11] vía web y por aplicaciones de terceros mediante las herramientas *Eutils* [12][13]. Las *Eutils* [12][13] nos devuelven la información que consultamos en ficheros *.xml.*

Por otro lado, KEGG (*Kyoto Encyclopedia of Genes and Genomes*)[10], proporciona los KEGG PATHWAYS. Estos PATHWAYS son una colección de grafos dirigidos en los que se representan interacciones moleculares y redes de regulación génica.

La herramienta dentro de *PCOPGene[2]*, que nos ayuda a buscar las **relaciones génicas** a partir de la información médica y biológica de unas bases de datos está compuesta de 3 partes claramente diferenciadas en el servidor local:

- Base de datos local con la información biomédica descargada del NCBI [11] y de Kegg [10] necesaria para encontrar **relaciones génicas**.

- Un aplicativo php que cruza la información biomédica de la Base de datos local, para encontrar genes relacionados con los genes de la microarray seleccionados por el usuario. Este aplicativo php consta de dos procesos, un primer proceso que realiza los cruces de las bases de datos y obtiene los **genes relacionados** y un segundo proceso interactivo que muestra los resultados y permite operar sobre ellos. Actualmente esta última fase es bastante precaria y su interactividad muy limitada, lo que supone un problema dada la gran cantidad de información que suministra.

- *El applet de PCOPGene*[2] contiene el *gene network* que permite trabajar de forma interactiva con los genes de cada *microarray*. Este applet permite llamar a la herramienta para obtener genes relacionados con genes de la microaray usando bases de datos biomédicas remotas. En la llamada a la herramienta se le pasará la selección de genes de la microrray con los que el usuario desea encontrar genes relacionados.

Uno de los inconvenientes de *POPGene[2]*, es que restringe el uso de la herramienta para buscar relaciones génicas a una selección de **genes de entrada** sobre el *gene network*, que tiene que realizar el usuario. Esto ocasiona los siguientes problemas:

- No permite obtener **relaciones génicas** de genes que no aparezcan en la *microarray* que se esté analizando. En algunos casos puede ser interesante buscar los **genes relacionados** con determinados genes de la *microarray*, pero en otros casos puede interesar buscar **genes relacionados** con genes no incluidos en la *microarray*, por ejemplo buscar los **genes relacionados** con genes vinculados a una patología o una función celular concreta. Y así encontrar los genes de la *microarray* relacionados con los genes vinculados a dicha patología.

- Un problema asociado a lo anterior es que un gen o una proteína sintetizada por el gen pueden tener múltiples nomenclaturas. Para estandarizar el uso de la aplicación *PCOPGene*[2], ha de aceptarse cualquier nomenclatura del gen o proteína cuando el usuario introduzca el nombre del gen manualmente.

- Otro problema asociado es que actualmente tampoco se permite obtener los genes relacionados con una patología o una función celular, es decir, no se puede comenzar la investigación sin partir como mínimo, de un gen de entrada. Sería interesante obtener directamente los **genes relacionados** con una patología o función celular solo introduciendo dicha palabra clave, por ejemplo, colon cancer o cell membrane.

El otro problema del aplicativo *PCOPGene*[2], es que al tener un volumen tan grande de información en la Base de datos local, son millones los genes potenciales a relacionar (al rededor de 8 millones). Esto provoca que cuando la herramienta para encontrar relaciones

génicas muestra sus resultados, muestra tanta información que resulta difícil de interpretar. Es por eso que la parte del aplicativo php que muestra los resultados y permite operar sobre ellos necesita ganar en operativilidad, interactividad, usabilidad y entendivilidad.

El tratarse de una base de datos local tan grande, también implica un coste de tiempo muy alto a la hora de realizar el cruce de las bases de datos para obtener las **relaciones génicas**. Teniendo en cuenta que se trata de una herramienta on-line, el coste de tiempo para la proceso de búsqueda de **relaciones génicas** ha de ser mínimo.

Por último, un inconveniente de *PCOPGene*[2] es que no tiene la información de la base de datos bien actualizada. Esta es una gran contrariedad, ya que la información científica en el campo de la biología y medicina se actualiza día a día con nuevos descubrimientos y experimentos de todo tipo.

Las limitaciones comentadas anteriormente, dificultan la investigación que realizan los usuarios mediante la herramienta de búsqueda de relaciones génicas de la aplicación *PCOPGene[2]*.

1.3. Objetivos

El objetivo principal del trabajo que nos abarca, es hacer más potentes las consultas que pueden realizar los usuarios para encontrar nuevas **relaciones génicas**. Para tener unas herramientas de consulta más potentes, necesitamos enriquecer la información de la base de datos local, descargando los datos necesarios de las bases de datos remotas de **NCBI** y **Kegg**.

Las bases de datos de **NCBI** y **Kegg** tienen un gran volumen de información, tendremos que encontrar el método de optimizar el almacenaje y consulta de todos estos datos en el servidor local. Existe la inminente necesidad de mantener la información de la Base de datos local actualizada, porque las bases de datos remotas del **NCBI** y **Kegg** actualizan y modifican casi diariamente sus datos. También tenemos que incrementar la operativilidad de la aplicación web, abriendo las consultas ahora limitadas a buscar genes relacionados con los genes de la *microarray*.

- Objetivos relacionados con el aplicativo php que realiza el cruce de las bases de datos biomédicas para encontrar nuevas relaciones entre los genes:

    - No restringir la búsqueda de relaciones génicas únicamente a los genes de la *microarray* seleccionados por el usuario, sino permitir que le lleguen al proceso de cruce nombres de genes externos a la *microarray* introducidos manualmente por el usuario.
    - Permitir todas las posibles nomenclaturas del gen o proteína en la entrada manual

de los genes. El proceso de cruce de las bases de datos, al recibir las diferentes nomenclaturas de un gen ha de encontrar el símbolo (nombre oficial) del gen referido.

- El proceso de cruce, ha de permitir buscar genes relacionados con términos específicos como el nombre de una patología o de una función celular. Es decir sin recibir **genes de entrada** para la búsqueda.
- El coste en tiempo de respuesta del actual proceso de cruce de las bases de datos es muy alto. Debemos encontrar un método adecuado para optimizar el cruce de información de la Base de datos local.

- Objetivos relacionados con el aplicativo php encargado de operar con las **relaciones génicas** encontradas:

  - En el proceso de operar con las relaciones génicas actual hace uso de las herramientas *Eutils* para descargar parte de la información que necesitamos y no tenemos en la base de datos local. Es necesario evitar el uso de las *Eutils* en el aplicativo php, ya que ralentiza la fase de visualización de los resultados.
  - El proceso para operar con las relaciones génicas, muestra las **relaciones génicas** entre los **genes de entrada y l**os **genes relacionados** con estos. Pero algunas de estas **relaciones génicas** también relacionan entre sí los **genes resultado**, con lo que hay que dotar al proceso de la capacidad suficiente para mostrar las **relaciones génicas** existentes entre grupos de **genes resultado** cuando lo solicite el usuario.
  - En el proceso de operar con las **relaciones génicas**, hay que buscar la forma optima de visualizar las listas de relaciones génicas. Superando todos los problemas y deficiencias actuales.
  - Actualmente en proceso de operar con las **relaciones génicas**, tenemos la opción de reordenar los listados de **relaciones génicas** por cualquier base de datos consultada pero no por los **genes relacionados**. Habilitar esta opción es primordial para estudiar todas las **relaciones génicas** encontradas en las que participa un mismo **gen relacionado**.
  - En definitiva conseguir una interfaz web muy intuitiva y amigable para los usuarios.

- Objetivos a la hora de redefinir la Base de datos local:

  - La base de datos local contiene un gran volumen de información y esto ralentiza significativamente la velocidad de respuesta del proceso de cruce. tenemos que reestructurar la Base de datos local para optimizar el acceso a la información contenida.

- Objetivos referentes a la actualización periódica automática de la Base de datos local:

- Conseguir una actualización trimestral robusta y que se amolde a las grandes inserciones de información que va a sufrir la Base de datos local.
- En el proceso de actualización, actualizar también el nombre de los genes en el fichero que lee *PCOPGene[2]* para construir la *gene network*.
- Controlar la correcta actualización de la base de datos en caso de caída del servidor local.
- Controlar la consistencia de la base de datos. Asegurar la correcta descarga de los archivos necesarios, en caso de que estos estén corruptos.

• Objetivos base en la adaptación del applet **PCOPGene-net** :

- Adaptar la aplicación java a los cambios realizados en el aplicativo php ya que el applet es el encargado de enviar los **genes de entrada** y otros datos de consulta al proceso de cruce de las bases de datos, así como recuperar los **genes relacionados** del apliativo php.
- No restringir la búsqueda de **relaciones génicas** únicamente a los **genes de entrada** de la *microarray* seleccionados por el usuario, sino permitir que el usuario entre nombres de genes manualmente o incluso no entre genes, sino palabras clave.
- Permitir al usuario que visualice los **genes relacionados** que obtenemos del aplicativo php en la *gene network* de la *microarray* que está analizando. Al marcar estos genes en el *gene network*, estos pueden ser utilizados por otras otras herramientas de PCOPGene[2].

1.4. Organización de la memoria

En este apartado explicaré como me he planteado afrontar la realización de mi proyecto y la solución a todos los problemas encontrados.

En la siguiente sección de la memoria, expongo los fundamentos teóricos necesarios para comprender los conceptos con los que trabajamos en el proyecto.

Seguidamente paso a explicar las tres fases en que se ha planificado el trabajo realizado.

En una primera fase me centro en el aplicativo php, primeramente en el proceso de cruce de las bases de datos biomédicas remotas para buscar **relaciones génicas** y posteriormente en el proceso de operar con las **relaciones génicas** encontradas. En la sección correspondiente a esta primera fase, primero explico el funcionamiento del proceso de cruce de bases de datos anterior y los problemas que me he encontrado. Luego expongo dos de las tres soluciones que he implementado, y el porque no han resultado satisfactorias. Finalmente, me centro en exponer la correcta solución a la que he llegado a partir de los problemas surgidos en las implementaciones anteriores. El resultado es una afectación tanto de la estructura de la Base de datos local como de la aplicación php, tanto en el proceso de cruces como en el de operar.

La segunda fase del proyecto se centra en la actualización de la Base de datos local. En una primera parte expongo los requisitos que me han planteado y que tiene que cumplir la actualización.

En siguiente lugar, explico en detalle el flujo de ejecución de la actualización y expongo de que manera he adaptado la actualización existente a la nueva estructura de la Base de datos local y como adaptar la estructura de dicha Base de datos local al gran crecimiento de información.

La tercera fase, se centra en añadir al applet **PCOPGene-net** las nuevas funcionalidades que son objetivo de este proyecto, y como recuperar desde el *gene network* los **genes relacionados**.

En la descripción de las respectivas fases, expongo como se han ido alcanzando los diferentes objetivos uno a uno y porque las opciones tomadas son las mejores posibles para alcanzar dichos objetivos.

La última sección es el informe técnico. La función del informe técnico es facilitar la reusabilidad, adaptavilidad a futuros desarrolladores.

2. Fundamentos teóricos

2.1. Bioinformática

La Biología computacional o bioinformática es la ciencia dedicada al estudio de los fenómenos biológicos de la microbiología molecular desde un punto de vista computacional, con el objetivo de ofrecer métodos robustos para la comprensión, simulación y predicción de comportamientos biológicos observados en los seres vivos.

Bioinformática es un campo de la ciencia en el cual confluyen varias disciplinas tales como: biología, computación y tecnología de la información. El fin último de este campo es facilitar el descubrimiento de nuevas ideas biológicas así como crear perspectivas globales a partir de las cuales se puedan discernir principios unificadores en biología. Al comienzo de la "revolución genómica", el concepto de bioinformática se refería sólo a la creación y mantenimiento de base de datos donde se almacena información biológica, tales como secuencias de nucleótidos y aminoácidos. El desarrollo de este tipo de base de datos no solamente significaba el diseño de la misma sino también el desarrollo de interfaces complejas donde los investigadores pudieran acceder los datos existentes y suministrar o revisar datos.

La bioinformática consiguió parte de la importancia que tiene actualmente con el nacimiento del Proyecto Genoma Humano (HGP). No se trataba simplemente de leer la secuencia de DNA del ser humano, ya que la parte práctica del HGP (obtener la secuencia con métodos analíticos de laboratorio) ya hace varios años que finalizó, pero la parte teórica, comprender el significado de 3.2 GB de información, dista mucho de estar completada. Ante esta gran cantidad de datos que generan las nuevas técnicas de biología molecular, *el reto fundamental de la bioinformática es adaptar y utilizar técnicas computacionales existentes, o crear nuevas, para poder extraer información útil de estos datos (Data mining)*. Esta información ha de servir para complementar y validar la ya existente basada en experimentos clásicos, para generar nuevos conocimientos sobre determinados procesos biológicos y también descubrir nuevos.

Combinada con las nuevas técnicas de biotecnología, la bioinformática permite identificar genes y proteínas causantes de enfermedades o de mecanismos de resistencia a antibióticos, de otra forma complicados de detectar. En estos métodos, permite analizar de forma extensiva y lo suficientemente amplia los genes implicados, recreando un modelo aproximado, y en consecuencia pudiendo intervenir en los procesos con fármacos más específicos, o incluso con nuevos paradigmas terapéuticos, como la terapia génica.

La bioinformática sería otro campo más de aplicación de técnicas computacionales si no fuera porque el campo sobre el que se aplican en este caso, la biología molecular, está experimentando un crecimiento exponencial. Con este panorama, el número y la complejidad

de los datos aumentan a un ritmo más alto que la capacidad de los ordenadores que los han de procesar, la necesidad de un fuerte desarrollo teórico y práctico en el entorno de las herramientas bioinformáticas se hace más que necesario. Otro de los problemas a la hora de tratar toda esta información es la gran cantidad de ruido que hay, al tratarse de datos obtenidos experimentalmente.

Una vez se han clasificado los datos obtenidos, el último paso en el análisis de los mismos consiste en extraer la información biológica subyacente a los resultados experimentales. Para ello, se han desarrollado numerosas herramientas que obtienen información a partir de bases de datos u ontologías biológicas y permiten sacar conclusiones finales referentes al experimento realizado. Una especialidad dentro de la biotecnología es la denominada ingeniería metabólica, una nueva disciplina que tiene por objetivo último la optimización de las redes génicas, el estudio de la interacción entre los genes, así como el desarrollo de software para su estudio.

2.2. Cluster

Los cluster de genes, son agrupaciones de genes por condiciones muestrales o experimentos con un comportamiento similar o semejantes en el valor de expresion. Es decir, podemos agrupar los genes en clusters estudiando la expresión que tiene dicho gen en una *microarray*, como respuesta a un experimento. Los genes que tengan una respuesta similar, serán agrupados en clusters.

2.3. Bases de datos estudiadas y utilizadas

El centro de datos de donde descargamos el 90% de la información que tenemos estructurada en nuestra base de datos local es el Centro Nacional para la Información Biotecnológica o *National Center for Biotechnology Information* (NCBI).

De toda la información del NCBI, escogeremos sólo la que nos es útil para nuestra aplicación:

Bases de Datos del NCBI	
Base de datos	**Información**
Gene	Base de datos con información básica del gen: *Taxonomy* (Especie a la que pertenece), Símbolo, Nombre, Cromosoma en el que se encuentra, Interacciones con otros genes, etc.
PubMed	Base de datos que contiene publicaciones en *papers* sobre experimentos e investigaciones en genes.
UniGene	Base de datos central cuya principal función es la unificación de los genes. Se empleará para la obtención de los GeneIDs (códigos de gen) que identifican unívocamente a cada gen.
OMIM	Base de datos que contiene patología de cada gen. Se empleará para cruzar genes por patologías.
Gene Ontology	Bases de datos con información acerca de la ontología de cada gen, de las cuales obtener que funciones realiza, en qué procesos intervienen y en qué componentes celulares se localizan.
Homologene	Base de datos con información acerca de genes homólogos. El empleo de esta información no será más que para descartar aquellos **genes relacionados** que sean homólogos.
Refseq (en nuestra base de datos será Map)	Para conocer el genoma en el que se encuentra un gen, así cómo su comienzo de manera que podamos indexar sus genes vecinos
Interactions	Para conocer los genes con los que interactúa un gen, así cómo el nombre que recibe dicha interacción

Tabla 2.2.1: Bases de datos remotas del NCBI

A parte de la información que facilita en NCBI desde el *ftp*, en este proyecto hemos

ampliado la base de datos local del IBB guardando los títulos de las publicaciones, de las patologías de cada gen y las diferentes nomenclaturas que tiene cada gen. Para esto hemos hecho uso, en la fase de la actualización (**3.2.**), de las herramientas que ofrece el NCBI como *webservice*, las *Eutils*.

Mediante las *Eutils* descargamos en formato *xml* los datos necesarios para el proyecto que nos abarca.

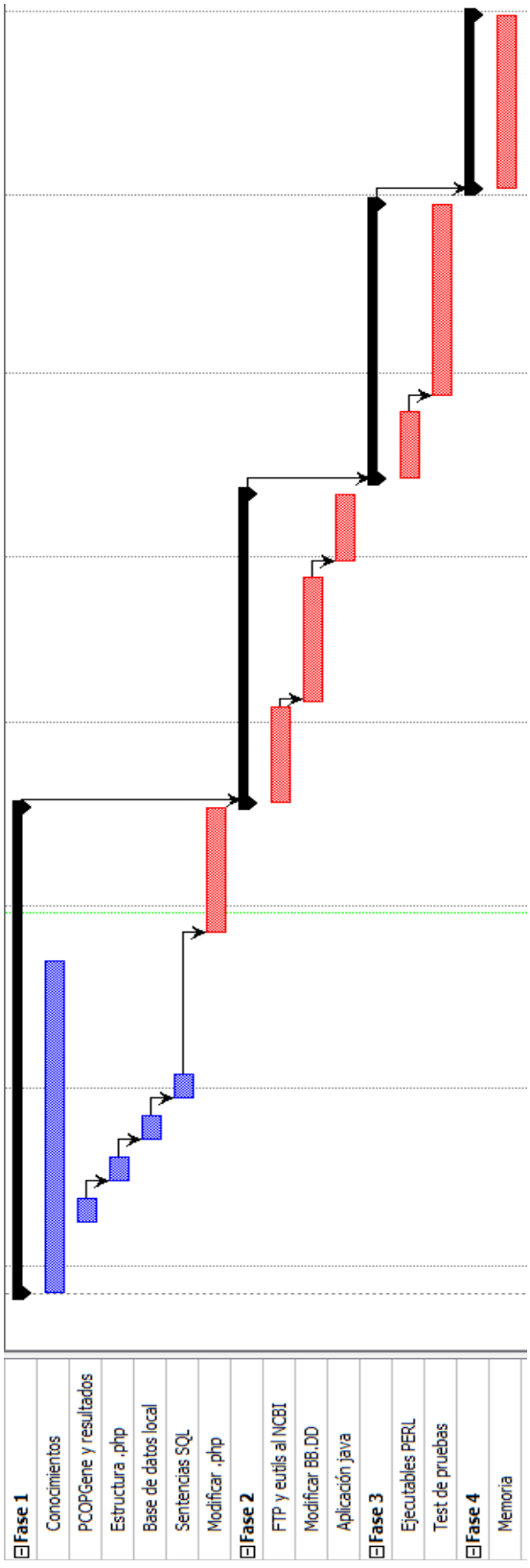
El 10% restante de la información la hemos extraído de KEGG (*Kyoto Encyclopedia of Genes and Genomes*). De este centro de datos extraemos los pathways que son una colección de mapas dibujados manualmente en los que se representan interacciones moleculares y redes de reacción.

- Metabolismos
- Información de Procesos Genéticos
- Información de Procesos del Entorno del gen
- Procesos celulares
- Enfermedades humanas

3. Fases

Planificación:

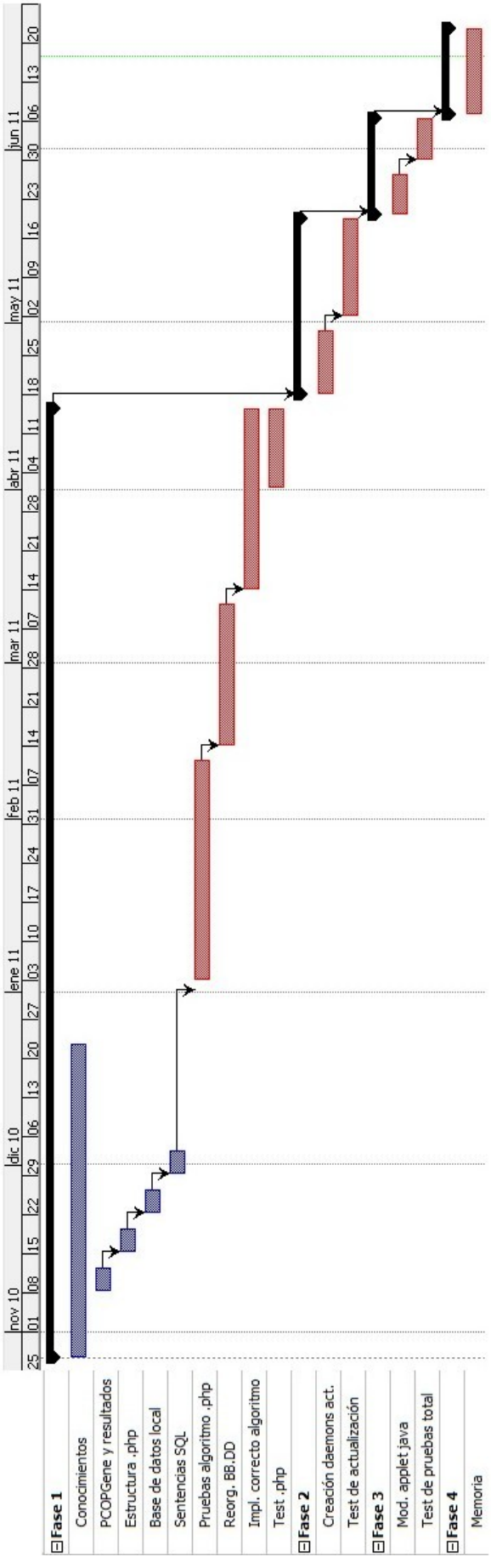
A continuación voy a exponer la planificación inicial que había planteado para el desarrollo de este proyecto.



NOMBRE	DURACIÓN	INICIO	TERMINADO
Fase 1	62,5 days	27/10/10 12:00	21/01/11 17:00
Conocimientos	40,5 days	27/10/10 12:00	22/12/10 17:00
PCOPGene y resultados	4,875 days	08/11/10 09:00	12/11/10 17:00
Estructura .php	4,875 days	15/11/10 09:00	19/11/10 17:00
Base de datos local	4,875 days	22/11/10 09:00	26/11/10 17:00
Sentencias SQL	4,875 days	29/11/10 09:00	03/12/10 17:00
Modificar .php	15,875 days	27/12/10 09:00	17/01/11 17:00
Fase 2	35 days	18/01/11 08:00	11/03/11 17:00
FTP del NCBI	13 days	18/01/11 08:00	03/02/11 17:00
Modificar BB.DD	10 days	04/02/11 08:00	25/02/11 17:00
Aplicación java	10 days	28/02/11 08:00	11/03/11 17:00
Fase 3	35 days	14/03/11 08:00	29/04/11 17:00
Ejecutables PERL	10 days	14/03/11 08:00	25/03/11 17:00
Test de pruebas	25 days	28/03/11 08:00	29/04/11 17:00
Fase 4	22 days	02/05/11 08:00	31/05/11 17:00
Memoria	22 days	02/05/11 08:00	31/05/11 17:00

Tabla 3.0.1.1: Planificación inicial

Seguidamente, voy a adjuntar la planificación real del proyecto, y a explicar los motivos por los cuales son distintas la planificación inicial y la real.



NOMBRE	DURACIÓN	INICIO	TERMINADO
Fase 1	122,5 days	27/10/10 12:00	15/04/11 17:00
Conocimientos	40,5 days	27/10/10 12:00	22/12/10 17:00
PCOPGene y resultados	4,875 days	08/11/10 09:00	12/11/10 17:00
Estructura .php	4,875 days	15/11/10 09:00	19/11/10 17:00
Base de datos local	4,875 days	22/11/10 09:00	26/11/10 17:00
Sentencias SQL	4,875 days	29/11/10 09:00	03/12/10 17:00
Pruebas algoritmo php	30 days	01/01/11 09:00	11/02/11 17:00
Reorg. BB.DD	19,875 days	14/02/11 09:00	11/03/11 17:00
Impl. correcta algoritmo	24,875 days	14/03/11 09:00	15/04/11 17:00
Test .php	11 days	01/04/11 08:00	15/04/11 17:00
Fase 2	22 days	18/04/11 08:00	17/05/11 17:00
Creación daemons act.	13 days	18/04/11 08:00	04/05/11 17:00
Test actualización	9 days	05/05/11 08:00	17/05/11 17:00
Fase 3	14 days	18/05/11 08:00	06/06/11 17:00
Mod. applet java	8 days	18/05/11 08:00	27/05/11 17:00
Test total pruebas	6 days	30/05/11 08:00	06/06/11 17:00
Fase 4	12 days	07/06/11 08:00	22/06/11 17:00
Memoria	12 days	07/06/11 08:00	22/06/11 17:00

Tabla 3.0.1.2: Planificación real

Si observamos la planificación con detenimiento, podemos ver que la fase que más ha cambiado respecto a la planificación inicial ha sido la que engloba las modificaciones referentes a la Base de datos local y a la aplicación php. Esto es debido a los diferentes problemas que he ido encontrando, los cuales he solucionado con éxito, pero después de concienzudos análisis, búsquedas de alternativas y implementación de diferentes soluciones posibles. Otra causa de que se retrasara esta fase, es que hemos ido viendo diferentes posibilidades que podía incluir en el proceso de operar con las relaciones génicas en el aplicativo php, como por ejemplo la de cruzar las relaciones génicas encontradas entre los **genes relacionados** obtenidos.

Prerequisitos en el servidor:

Servidor	Cliente
• Sistema de Gestión de Bases de Datos (mySQL, open source) • Java • Interpretes de Perl, con modulo de tratamiento de .xml. • Servidor web para plataformas Unix (Apache, open source, que ya tiene incluido un interprete de php) • Óptima conexión a Internet • Sistema operativo Linux, con permisos para ejecutar cron desde cualquier usuario. • Espacio disponible en disco para la posterior ampliación de la base de datos.	• Acceso a Internet • Máquina virtual de java • Explorador de Internet • Java

Tabla 3.0.2.1: Prerequisitos del sistema

Como podemos observar no son unos requisitos extremadamente costosos, ni por parte del servidor, ni por parte de los usuarios que quieran disfrutar de las posibilidades que ofrece este proyecto.

3.1. Optimizar aplicativo php

3.1.1. Visión general del aplicativo existente

En esta sección explicaré extensamente las fases que he ido recorriendo durante la realización del proyecto. Expondré una visión general de las dificultades que he encontrado y las posibles soluciones hasta encontrar la correcta. También, los cambios que he aplicado en los diferentes aplicativos y los resultados que justifican la versión final del proyecto.

Como ya hemos visto anteriormente (apartado **1.2.**), este proyecto consta de tres partes muy diferenciadas. En primer lugar, un aplicativo **php** donde podemos ver las **relaciones génicas** esperadas a partir de las consultas lanzadas por los usuarios. En segundo lugar, una base de datos local con toda la información necesaria para el estudio de los genes. Y, por último, applet *PCOPGene-net* mediante el cual lanzamos nuestras consultas para realizar el cruce de información al aplicativo php.

En una primera fase del proyecto, me he centrado en el estudio del aplicativo php, en el

flujo de información del algoritmo, y cómo solucionar los problemas que presentaba (comentados en la sección de objetivos **1.3**).

Las consultas se realizan seleccionando una serie de **genes de entrada**. Podemos realizar la consulta de todos los **genes de entrada** que hemos seleccionado (lo llamaremos operación **ADN**) o por cada uno de los **genes de entrada** que hemos seleccionado (operación **OR**).

En las consultas, podemos escoger varias opciones:

- **Pubmed:** Base de datos que contienen publicaciones en *papers* sobre experimentos e investigaciones sobre genes.
- **GO:** Bases de datos con información acerca de la ontología de cada gen, de las cuales obtener qué funciones realiza, en qué procesos intervienen y en qué componentes celulares se localizan.
- **MIM:** Base de datos que contienen la patología de cada gen. Se empleará para cruzar genes por patologías.
- **Map:** Encontramos los genes "vecinos" de un determinado gen.
- **Kegg:** Colección de mapas dibujados manualmente en los que se representan interacciones moleculares y redes de reacción, *pathways*.
- **Interactions:** Descripción de las interacciones entre dos genes y su resultado.

También tiene unas opciones de selección de dominio para restringir la muestra total de **genes relacionados**:

- All genes: total de genes de la base de datos.
- **Same cluster:** genes del mismo *cluster* que los **genes de entrada.**
- **Genes in the microarray:** genes en la misma *microarray* que los **genes de entrada.**
- **Genes with correlations:** genes con un factor de relación con los **genes de entrada.**
- **Genes selected:** sobre los mismos **genes de entrada.**

Este aplicativo constaba de dos archivos: **cruces.php** y **mostrar.php**.

En **cruces.php**, se generan dos archivos, como resultado a las consultas a la base de datos local con los **genes de entrada** y las opciones escogidas, un **.txt* y un **.xml*. En el **.txt* guardaba todos los genes que tenían resultados comunes, dentro del dominio, según la selección de lo que queremos comparar. En el **.xml*, almacena todas las **relaciones génicas** y por cada una todos los genes de la base de datos comunes a las opciones seleccionadas y relacionadas con los **genes de entrada**.

En **mostrar.php**, estos dos archivos generados anteriormente, se recorren y se insertan en una tabla intermedia de la base de datos. A la hora de imprimir los resultados por pantalla, se recorre esta tabla intermedia que almacenaba los identificadores de los genes y los identificadores de cada opción escogida en la consulta. Si se necesitaba algún dato más a mostrar, como por ejemplo podían ser publicaciones relacionadas con alguna interacción, se consultaba de nuevo la base de datos para encontrar la información necesaria.

En esta aplicación, encontrábamos algunos errores en el momento de imprimir los resultados que cumplían las condiciones del usuario, e inconvenientes en cuanto a tiempo de respuesta:

- En algunas consultas, mostraba una serie de **genes relacionados** sin estar relacionados con ninguna **relación génica**.
- Observamos que imprimía filas en blanco, debido a las **relaciones génicas** que se encontraban en el archivo **.xml*, y no tenían gen común en el archivo **.txt.*
- Esta aplicación permitía ordenar las **relaciones génicas** una vez mostrados, recorriendo de nuevo los archivos que había generado y volviendo a cargar la página. Esto ralentizaba mucho la ordenación.
- Nos dimos cuenta que la ordenación se ejecutaba por todas las opciones de **relaciones génicas** menos por **genes relacionados**.
- Con todo el cálculo y generación de archivos, resultaba una aplicación bastante lenta, cuando lo que se requiere en un aplicativo *on-line* es la velocidad al cargar los resultados.

Con estos inconvenientes, decidí implementar un nuevo algoritmo para solucionar estos problemas y mejorar el tiempo de respuesta del aplicativo. No resultó tarea fácil debido a la gran cantidad de información que albergaba la base de datos local, con lo que implementé un total de tres algoritmos diferentes hasta que encontré la solución definitiva. Estos tres algoritmos, los he implantado en el servidor de pruebas del IBB, *Devresearch*[1].

3.1.2. Algoritmos implementados

En este apartado, explicaré los tres algoritmos que implementé. Los dos primeros fracasaron en cuanto a tiempo de respuesta respecto del aplicativo anterior. El tercero, es la solución que actualmente está en uso y que mejora el coste en tiempo frente al anterior aplicativo.

En las tres soluciones que explicaré a continuación, he prescindido de implementar dos archivos para separar el cálculo y la visualización de los resultados. En mi algoritmo

1 http://devresearch.uab.es/

implemento todo junto, de esta forma no necesito guardar los resultados en ficheros intermedios.

3.1.2.1. Consultas cruzadas

Aquí presento la primera solución implementada. He decidido basar todo el cálculo de los resultados en consultas lanzadas a la base de datos local, es decir, dependiendo de las opciones que seleccione el usuario, el aplicativo de cruces php va creando la consulta a la base de datos y, de aquí, generaré las **relaciones génicas** correspondientes.

En una primera parte, selecciono toda la información que queremos saber de los **genes de entrada**. Por ejemplo, si queremos saber los genes comunes a los **genes de entrada** según las publicaciones, lo primero que hago es seleccionar las publicaciones relacionadas con los **genes de entrada**. Por cada una de las **relaciones génicas** obtenidas en la consulta anterior, selecciono los genes que tienen relación con las publicaciones, seleccionadas anteriormente, y muestro los **genes relacionados** y sus **relaciones génicas**.

En el momento de mostrar las **relaciones génicas**, simplemente tengo que imprimir por pantalla lo que me devuelve la consulta SQL[15], ya que en ella está todo lo que el usuario quiere estudiar de un grupo de genes en concreto, respecto del total.

A simple vista, parece una opción muy rápida y fácil de implementar. El problema es cuando tenemos varias opciones seleccionadas, aquí entran las consultas cruzadas.

Las consultas anidadas tienen la siguiente forma:

SELECT campo1,campo"... **FROM** tablaName1 **WHERE** cond1 **OR** cond2 **IN**

(**SELECT** campo1,campo"... **FROM** tablaName2 **WHERE** cond1 **OR** cond2 **IN**

(**SELECT** campo1,campo"... **FROM** tablaName3 **WHERE** cond1 **OR** cond2 **IN**...))

Esto implica que, por cada opción que escojamos, hay que crear una consulta SQL que discrimine **genes relacionados**. Los genes que nos devuelva la primera consulta -de todas las consultas cruzadas- serán los realmente comunes a los **genes de entrada** seleccionados, por tanto los genes que tengamos que mostrar.

Si observamos con atención las consultas anidadas, vemos que en las condiciones de cada consulta, todas las que están después de la palabra **WHERE**, tenemos la palabra **OR**.

Esto quiere decir que, los genes que seleccionamos en cada consulta, tienen que cumplir, como mínimo, una de las condiciones que estamos pidiendo. Un ejemplo de consulta a la base de datos, pidiendo los genes comunes a unas publicaciones en concreto, es el siguiente:

SELECT GeneID **FROM** gene2pubmed **WHERE** PMID = 1 **OR** PMID = 2 **OR** PMID = 3 **OR**...

Con esto, quiero decir que por cada condición, tiene que recorrer la tabla entera de publicaciones y mirar qué genes son comunes. Si a esta consulta le añadimos otras consultas anidadas, dependiendo de las opciones que hayamos escogido, el cómputo en procesamiento de la CPU resulta excesivamente alto y esto se refleja en un tiempo de respuesta muy grande, por lo tanto, se colapsa la base de datos. Como consecuencia de dicho colapso, más de un usuario no podría estar consultando datos al mismo tiempo. Ésto resultaría nefasto para una aplicación *on-line* en la que pueden acceder mucho usuarios al mismo tiempo desde cualquier parte del mundo.

3.1.2.2. Almacenando resultados en vectores temporales

Viendo los problemas de cómputo que estaba ocasionando en la base de datos local, decidí implementar una solución similar a la que estaba implantada en un principio: almacenar los resultados en vectores temporales. El motivo fue porque, el hecho de recorrer los vectores, resulta más rápido debido a que están en memoria principal y no en disco, como es el caso de los ficheros intermedios que se creaban en la anterior aplicación.

En este algoritmo, por cada opción que ha seleccionado el usuario, genero dos vectores. Uno contendrá los resultados de los **genes de entrada** a esa opción y en el otro, los genes comunes a los resultados de dicha opción. Como voy evaluando las opciones secuencialmente, en cada opción escogida tendré que cruzar los genes comunes de la opción anterior con los genes comunes de la opción que estoy evaluando para tener el vector resultado de los genes comunes a todas las opciones, es decir:

Opción1: Pubmed

Vector relaciones génicas de publicaciones a los genes de entrada:

{PMID1, PMID2, PMID3, PMID4}

Vector genes comunes:

{Gen1, Gen2, Gen3}

Opción2: Interactions

Vector relaciones génicas de interacciones a los genes de entrada:

{Inter1, Inter2}

Vector genes comunes:

{Gen1, Gen4, Gen6}

Vector de genes comunes total:

{Gen1} (el vector de genes comunes resultado será el cruce del vector de genes comunes de cada opción)

Al obtener el vector total de **genes relacionados**, recorro los vectores resultado de cada opción con la información referente a los **genes de entrada**. Después, lanzo consultas simples a la base de datos para coger los datos necesarios como pueden ser, en el caso del ejemplo anterior, títulos de las interacciones, etc.

Con esta solución, la base de datos local quedaba totalmente libre para poder resolver consultas de varios usuarios al mismo tiempo.

Aquí me encuentro con el problema de que no todos los **genes relacionados** tienen el mismo número de entradas para cada **relación génica**, es decir, puedo tener el siguiente caso:

Gen	Pubmed	Interactions
Gen1	PMID1	Inter1
Gen1		Inter2
Gen1		Inter3

*Tabla 3.1.2.2.1: Muestra de diferente numero de **relaciones génicas***

Como podemos ver en el ejemplo anterior, el **Gen1** tiene una entrada para *Pubmed* común a los **genes de entrada**, pero tiene tres entradas para *interactions*. Hay que saber cuál de las **relaciones génicas** tiene el mayor número de entradas porque esa será la que controle el número de filas que tendrá como resultado el **Gen1**.

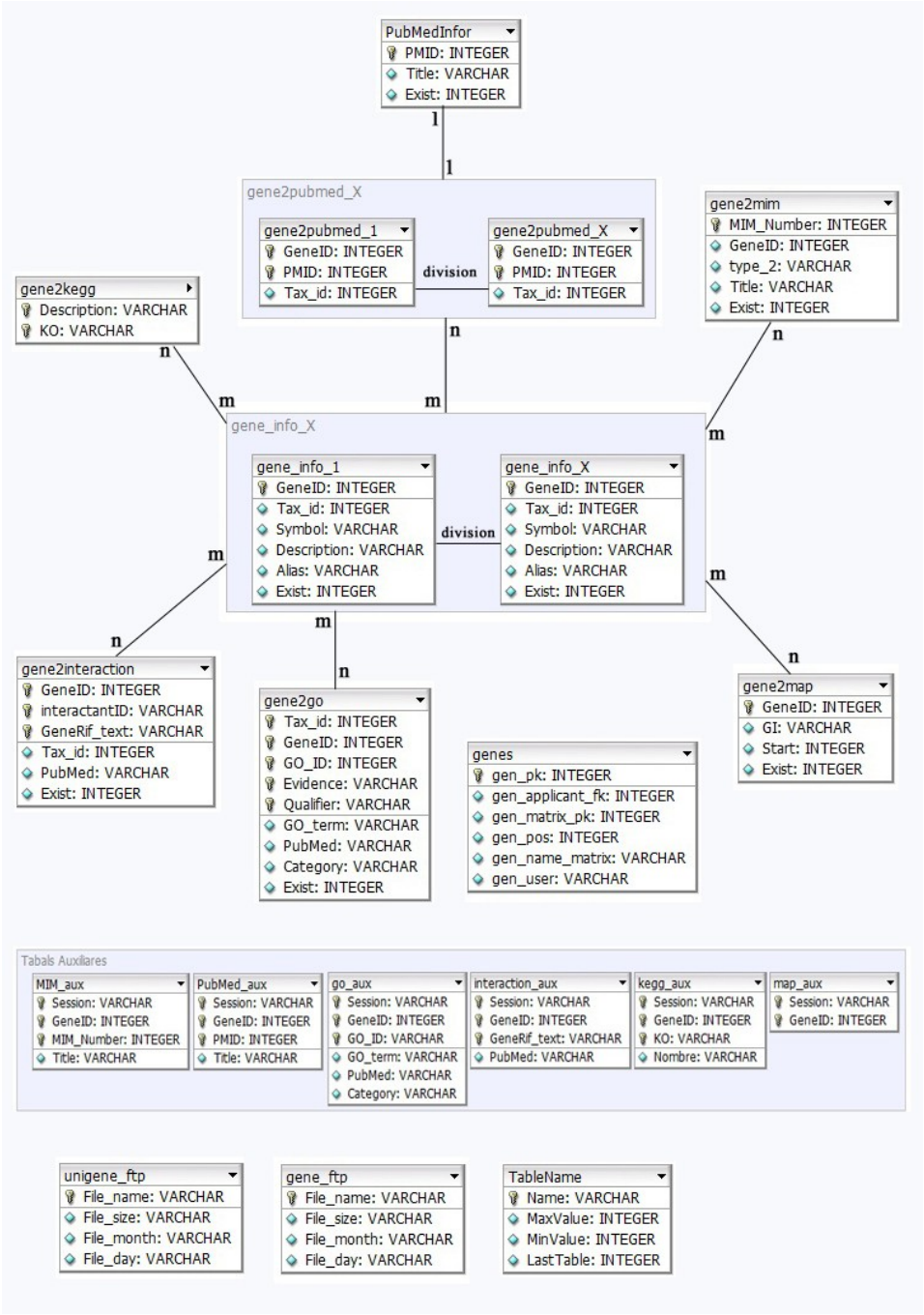
Este algoritmo parece bastante factible aunque el inconveniente más grande viene dado al hecho de que no lanzamos consultas anidadas a la base de datos. Tenemos que solventarlo recorriendo los vectores en bucles, como consecuencia, cuantas más opciones tenga el usuario marcadas, más bucles tendremos que ejecutar y esto eleva mucho la complejidad del algoritmo, -por lo general la complejidad de los bucles anidados es O(n^2) donde n es el número de bucles anidados- cosa que vemos reflejada en un tiempo de respuesta más alto que la aplicación original.

3.1.2.3. Uso de tablas auxiliares

Este algoritmo soluciona todos los problemas del aplicativo original y, actualmente, es el que reside en uso en el servidor del IBB.

Tras valorar los errores cometidos en las otras dos implementaciones antes explicadas, me dí cuenta de que era un problema más grande de lo que en un principio había pensado. No iba a encontrar la solución simplemente modificando el algoritmo de cruce de información. Llegados a este punto, decidí empezar a estudiar la estructura de la base de datos para poder aplicar modificaciones en ella.

Actualmente, la estructura de la base de datos está creada bajo un motor gestor de bases de datos MySQL [15] y es la siguiente:



Img. 3.1.2.3.1: Estructura de la base de datos

3.1.2.3.1. Mejorar tiempo de respuesta de acceso a la BBDD

Podemos ver que todas las tablas están relacionadas a través de una tabla principal que es la de gene_info. La tabla de gene_info es la más usada en esta aplicación, y tiene cerca de 4 millones de registros. Junto con ella, otra de las tablas muy consultadas por los usuarios es la de gene2pubmed, con unos 3,5 millones de registros. Son tablas muy grandes y tienen un riesgo alto de crecer excesivamente en las actualizaciones. Este hecho influiría notablemente

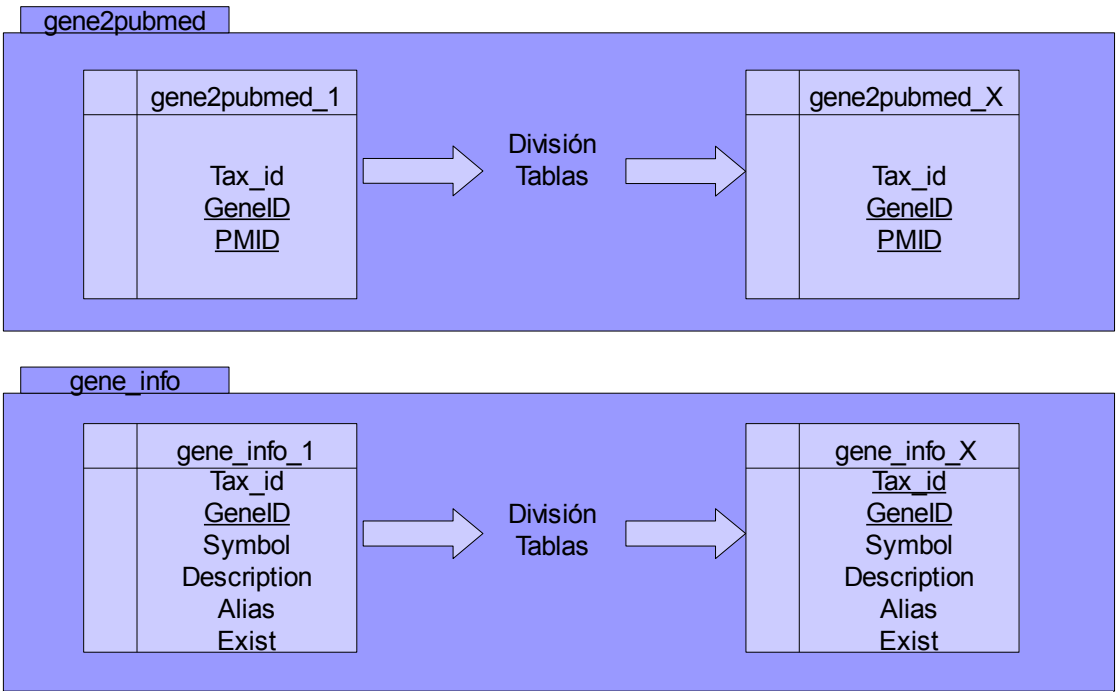
en el rendimiento de la aplicación y es un problema a solucionar porque hay que adaptar el aplicativo a los grandes crecimientos de información.

En la base de datos de la aplicación anterior, si queremos consultar la información de un gen y buscamos por **GeneID** en la tabla **gene_info** de 4 millones de registros, el resultado de la consulta tarda alrededor de **3,30** segundos en devolver el resultado. En la tabla de **gene2pubmed** de 3,5 millones, si consultamos todos los genes de un cierto PMID, nos encontramos con un tiempo de respuesta de **1,50** segundos de media.

Estos tiempos son excesivamente exagerados si tenemos en cuenta que podemos encontrar miles de **genes relacionados** con los **genes de entrada** ya que, para encontrar el **symbol** y la **description** de un gen, tenemos un coste de tiempo de **3,30** segundos de media.

Tras estudiar el tiempo, decidí proponer de reestructurar la base de datos local dividiendo las dos tablas -que son las más grandes y de mas peticiones de acceso- equitativamente en varias tablas mediante el campo clave de cada una, que es un número entero positivo. El número de registros de cada tabla lo mantengo alrededor de unos 200.000 elementos con un tiempo de respuesta de **0,0003** segundos de media. Crear cada tabla con un número menor que 200.000, no mejora el tiempo de respuesta porque el tiempo mínimo de acceso a una tabla es de **0,0003** segundos.

Con esta división, las tablas quedan estructuradas de la siguiente forma:



Dib. 3.1.2.3.1.1: Estructura de tablas divididas

Por tanto, he decidido no dividir todas las tablas de la base de datos porque el tiempo

de respuesta de todas las tablas, exceptuando las que hemos dividido (gene_info_X y gene2pubmed_X), es óptimo al rededor de **0,0003** segundos de media. Actualmente, si implementara un sistema de tablas divididas para toda la base de datos, bajaría el rendimiento ya que no conseguiría que el tiempo de respuesta de una tabla fuese menor de **0,0003** segundos.

Con esta estructura de tablas hay que implantar algún método para que la cantidad de tablas existentes en la base de datos, sea transparente al aplicativo de cruces php. Es decir, el algoritmo de cruces de información, tiene que adaptarse automáticamente al número de tablas que tenga la base de datos.

Para hacer que la estructura de la base de datos sea transparente a la aplicación php, he creado una tabla que guarda los nombres de todas las tablas de *gene_info* y de *gene2pubmed* con el siguiente formato:

- gene_info ⇨ gene_info_X (donde X será un número entero positivo identificador de cada tabla)

- gene2pubmed ⇨ gene2pubmed_X (donde X será un número entero positivo identificador de cada tabla)

Esta tabla, la he llamado TableName y tiene la siguiente estructura:

TableName	
Campos	**Información**
Name	Nombre de la tabla (gene_info_X o gene2pubmed_X)
MaxValue	Mayor valor de este campo para la tabla
MinValue	Menor valor de este campo para la tabla
LastTable	Identificamos si es la última tabla o no.

Tabla 3.1.2.3.1.2: Estructura TableName

El uso de esta tabla es para cuando vayamos a consultar un gen o una publicación usando su GeneID o su PMID, los identificadores de cada uno. Primero consultamos *TableName* y sacamos todos los registros que tengan como nombre *'%gene_info_%'* o

'%gene2pubmed_%'. Recorremos el resultado de la consulta y por cada tabla miramos si el gen o publicación que queremos consultar está dentro del rango de *MaxValue* y *MinValue*, en caso de que nos dé positivo, ya tenemos el nombre de la tabla donde tenemos que ir a consultar la información que queremos.

Con esta división, estamos gastando un tiempo de:

- Seleccionar todas las tablas *'%gene_info_%'* *o* '%gene2pubmed_%' : **0,0003 seg**.
- Recorrer la consulta para comparar si está en la tabla el identificador: **0,000286 seg** (en el peor de los casos siendo la última tabla)
- Seleccionar, de la tabla correcta, el registro deseado: **0,0003 seg.**
- Tenemos un total de **0,000886 seg** de media en la consulta de la información referente a un identificador.

Otra optimización que he implementado, para la reducción de tiempo de consulta a la base de datos, es la creación de índices[23] para todas las tablas. Los índices se crean sobre un campo que utilizaremos en las clausulas de las consultas y agilizan la búsqueda en las tablas de la misma forma que un índice de un libro.

Con todas estas optimizaciones, estamos reduciendo drásticamente el tiempo de consulta. Ahora queda plantearse la optimización del algoritmo de cruces del aplicativo php que trataremos en la siguiente sección.

3.1.2.3.2. Optimización del algoritmo de cruces php (tablas auxiliares)

Una vez solucionado el asunto con la velocidad de respuesta de las consultas a la base de datos, me dispongo a solucionar el problema sobre el tiempo de espera en el cruce de información del aplicativo php[16].

El problema de los otros dos métodos que he implementado es la gran cantidad de información que hay que consultar varias veces. Mi algoritmo muestra información, linea por linea, es decir que, una vez tiene los **genes relacionados**, vuelve a seleccionar de la base de datos, los resultados comunes a las opciones escogidas por el usuario. Con esto me refiero a que, si tenemos un **gen relacionado**, miramos qué opciones tiene marcadas el usuario. Por cada opción, consultamos a la base de datos las **relaciones génicas** comunes a los **genes de entrada**. En este bucle, vamos recorriendo uno a uno los resultados a mostrar y los vamos imprimiendo en pantalla linea por linea. De esta forma, es imposible imprimir en pantalla lineas vacías.

Hay que encontrar un modo de guardar las **relaciones génicas** que ya hemos

2 http://www.ignside.net/man/mysql/indices.php

3 http://www.tufuncion.com/optimizar_mysql

consultado de los **genes de entrada**, para no tener que volver a consultar sobre toda la base de datos. Ya hemos visto que, guardar la información en ficheros o en vectores temporales, no es una buena solución. Con estos inconvenientes decidí crear una tabla auxiliar por cada tabla consultada de esta forma:

Tablas auxiliares	
Tablas consultadas	**Tablas análogas auxiliares**
gene2go	**go_aux**
gene2interaction	**interaction_aux**
gene2mim	**mim_aux**
gene2map	**map_aux**
gene2pubmed	**pubmed_aux**
gene2kegg	**kegg_aux**

Tabla 3.1.2.3.2.1: Tablas auxiliares

En cada tabla auxiliar, guardamos la información que queremos mostrar, información que necesitamos para discriminar resultados y el GeneID que se relaciona:

Estructura de campos de las tablas auxiliares					
go_aux	**interaction_aux**	**mim_aux**	**map_aux**	**pubmed_aux**	**kegg_aux**
Session	Session	Session	Session	Session	Session
GeneID	GeneID	GeneID	GeneID	GeneID	GeneID
GO_ID	GeneRif_text	MIM_Number		PMID	KO
Go_term	Pubmed	Title		Title	Nombre
PubMed					
Category					

Tabla 3.1.2.3.2.2: Campos de tablas auxiliares

Como son tablas extremadamente pequeñas, resulta mucho más rápido consultar las **relaciones génicas** comunes a los **genes relacionados** directamente a estas tablas antes que a las tablas de la base de datos con toda la información. De esta forma, para cada **gen relacionado,** iremos a las tablas auxiliares y seleccionaremos la información que hemos decidido insertar en estas tablas, dependiendo de los filtros que el usuario haya aplicado a la consulta.

Si observamos con detenimiento la estructura de las tablas auxiliares, podemos observar un campo común a todas que es *"Session".* En este campo guardamos la fecha actual y el momento de la consulta con el siguiente formato: *"dd-mm-yy_hh:mm:ss"*. La finalidad de este campo es poder tener varios usuarios conectados a la vez a nuestra aplicación php, es decir, sirve para controlar que el aplicativo php sea multiusuario. Por ejemplo, imaginemos que tenemos el usuario **A** que selecciona todos los **Pubmeds** comunes a un **Gen1**, mientras este usuario está visualizando su información, se conecta un usuario **B**, consultando los **Pubmeds** comunes a un **Gen2**. Si no guardáramos la *"Session"* cada vez que un usuario hace su consulta, el usuario **B** estaría viendo sus resultados y algunos resultados de la consulta del usuario **A**, por ejemplo **genes relacionados** comunes a **Gen1** y **Gen2** aunque con Pubmed diferentes. Esto es debido a que en esta tabla sólo guardamos los resultados, que han superado el filtro deseado en la consulta, y seleccionamos los resultados por *"Session"* y GeneID, pero no volvemos a comprobar que sean comunes a los **genes de entrada**.

Tras solucionar el inconveniente de las sesiones, tenemos que plantear otro problema: las tablas auxiliares han sido creadas para acotar la información de las **relaciones génicas** comunes a los **genes de entrada**. Con esto quiero decir que, éstas no pueden contener información indefinidamente, ya que si albergaran información sin borrarla se convertirían en unas tablas muy grandes y resultaría más ineficiente que consultar a las tablas originales. Para esto, propuse dos soluciones:

- Por una parte, borrar la información referente a la sesión de cada usuario, cada vez que este cerrara el aplicativo php encargado de operar con las **relaciones génicas**. Esto implicaba tener que cargar una página alternativa para ejecutar el borrado de las tablas auxiliares, ya que el evento de cerrar o descargar el aplicativo php se produce en el ordenador del usuario y no podemos acceder a la base de datos desde java script por seguridad.

- Por otra, crear un *daemon* que se lanzara diariamente para borrar todo el contenido de las tablas auxiliares, independientemente de la sesión. Como inconveniente, si en el momento de borrar las tablas hay algún usuario conectado, las funcionalidades que tiene el aplicativo php para cruzar las **relaciones génicas** de forma *on-line*, no darán resultados (estas funcionalidades serán explicadas más adelante en el apartado **3.1.2.3.3.1**). La solución para el usuario es volver a realizar la consulta.

Así que decidimos implementar la segunda opción porque consideramos que era mejor para concentrar toda la acción de borrado de las tablas auxiliares en un cierto momento del día cuando pocos usuarios resultasen afectados. Estamos hablando de milisegundos ya que el borrado de las tablas es casi inmediato. De esta forma se evita que todos los usuarios sufran el borrado de las tablas auxiliares cada vez que cerrasen el aplicativo php donde operamos con las **relaciones génicas**.

Podemos ver que en el diseño de la base de datos hemos incluido el título de los **Pubmeds** y de **Mim** en nuestra base de datos local. En el aplicativo anterior se consultaba mediante las *Eutils,* pero ralentizaba mucho el mostrado de relaciones génicas en la aplicación php, ya que en una consulta que tenga miles de resultados para cada uno de ellos tiene que acceder al *webservice* gratuito del NCBI, y descargar el título. Para esto, he añadido la información necesaria en la base de datos.

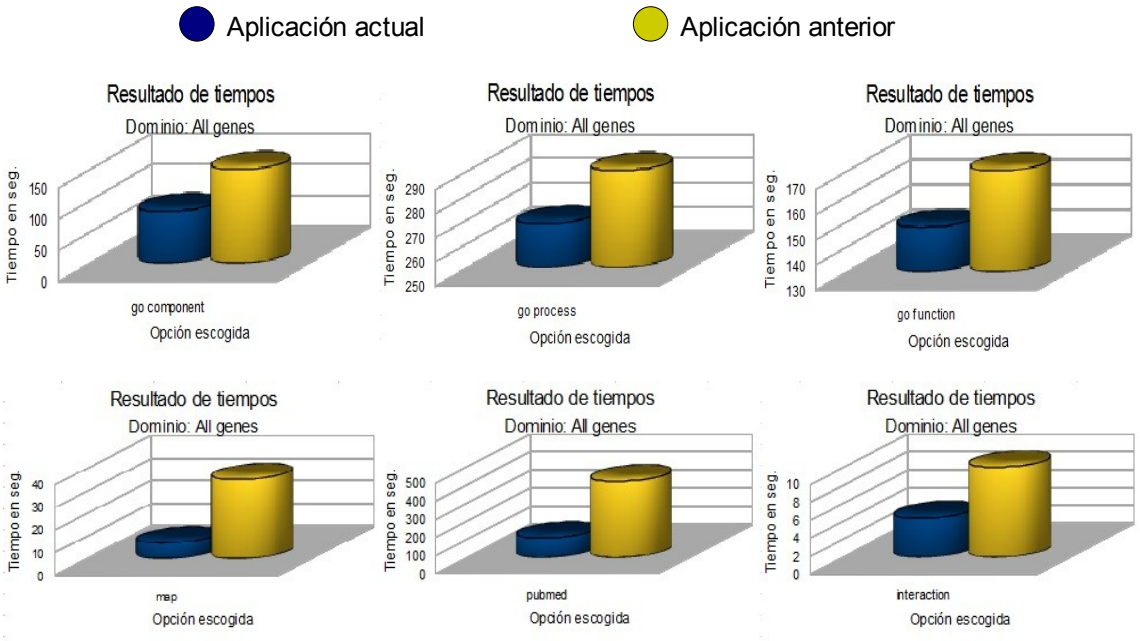
- En el caso de **Pubmed**, he añadido en una nueva tabla, llamada **PubmedInfo**, que tendrá el título de cada Pubmed y su PMID. En este caso, creamos una tabla debido al gran numero de información que tenemos en Pubmed y elevado riesgo de redundancia en la información.
- En el caso de **MIM**, he añadido los títulos en la misma tabla, ya que es una tabla muy pequeña y no alberga excesiva redundancia en los datos.
-

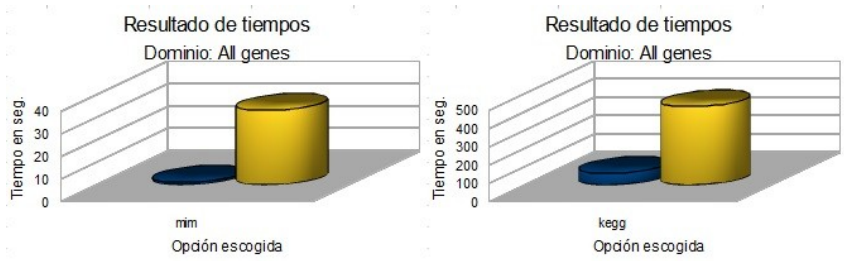
Una vez resuelto estos inconvenientes, tenemos el aplicativo php operativo.

Ahora ya podemos ver una comparativa de tiempos con el aplicativo anterior y observar la mejora que hemos conseguido:

La siguiente muestra está basada en el **gen de entrada** TP53, uno de los genes más estudiados. Con esto quiero decir que vamos a obtener muchos resultados y se van a realizar movimientos de grandes volúmenes de información de los que vamos a valorar el tiempo de respuesta en las dos aplicaciones.

He realizado las pruebas con los dominios de “All genes”, “Same cluster” y “Genes in the microarray”. Quedan dos dominios por mostrar: “Genes selected” y “Genes with correlations”. No he realizado el estudio de tiempos de respuesta en estos casos porque son muestras de genes con pocas **relaciones génicas** y la mejora en estos casos es despreciable.





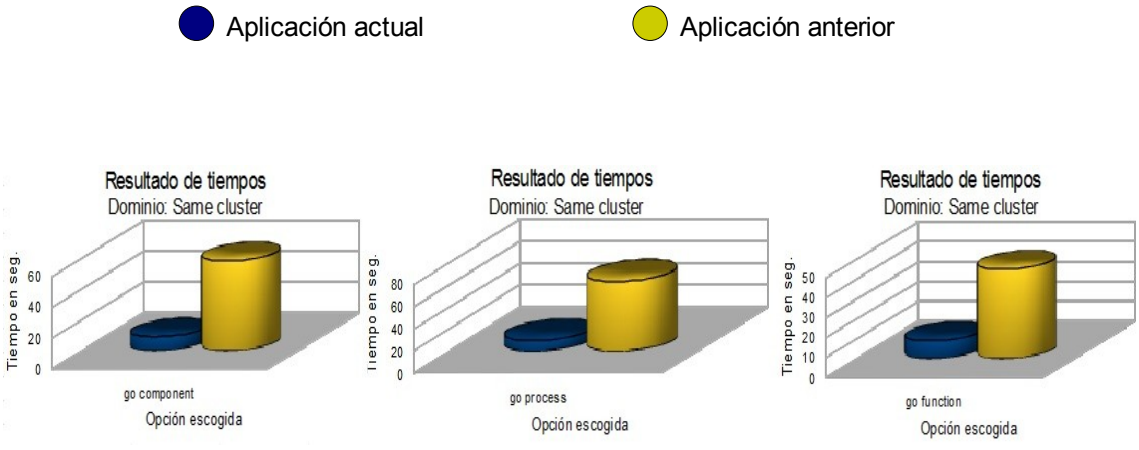
Img. 3.1.2.3.2.1: Resultados tiempos "All genes"

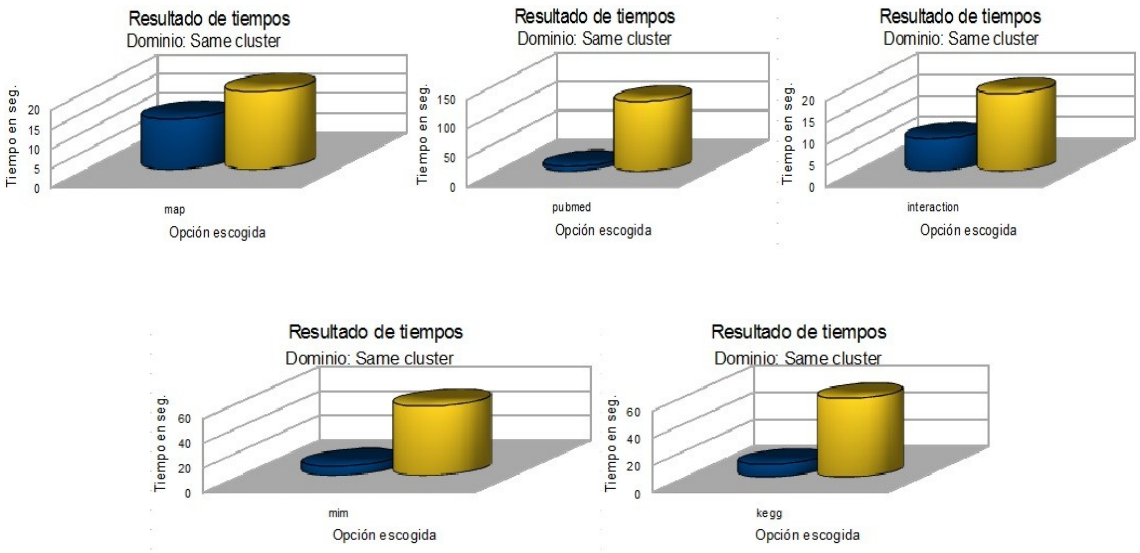
En el caso anterior, estamos mostrando los resultados de tiempo obtenidos. Por una parte, para realizar esta recogida de datos, hemos seleccionado todas las opciones posibles de búsqueda de información. Por otro lado, el dominio de “All genes”, buscará resultados por todos los genes que tenemos en la base de datos.

Seguidamente mostraré los porcentajes de mejora de cada opción:

Op. escogida	% mejora
map	80,30%
interaction	56,62%
pubmed	74,71%
kegg	84,73%
mim	96,84%
go function	13,10%
go process	7,52%
go component	44,89%

Tabla 3.1.2.3.2.3: Porcentajes mejora "All genes"





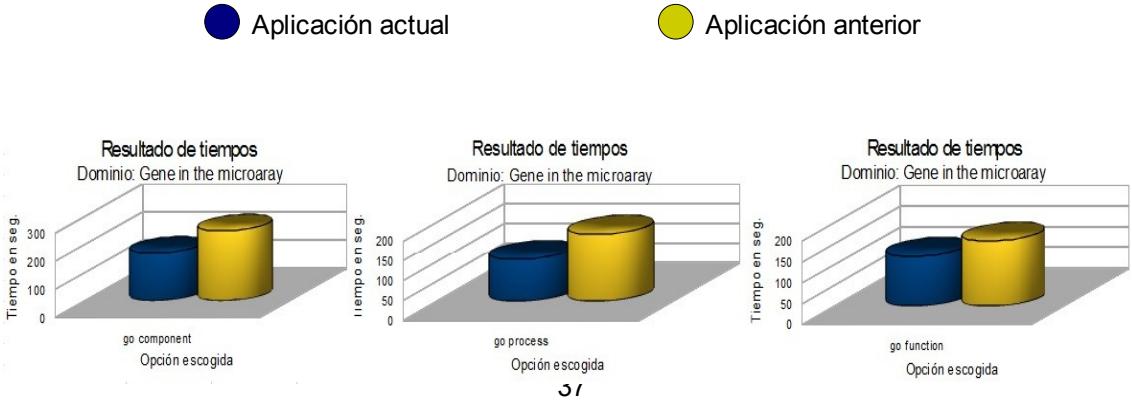
Img. 3.1.2.3.2.2: Resultados tiempos "Same Cluster"

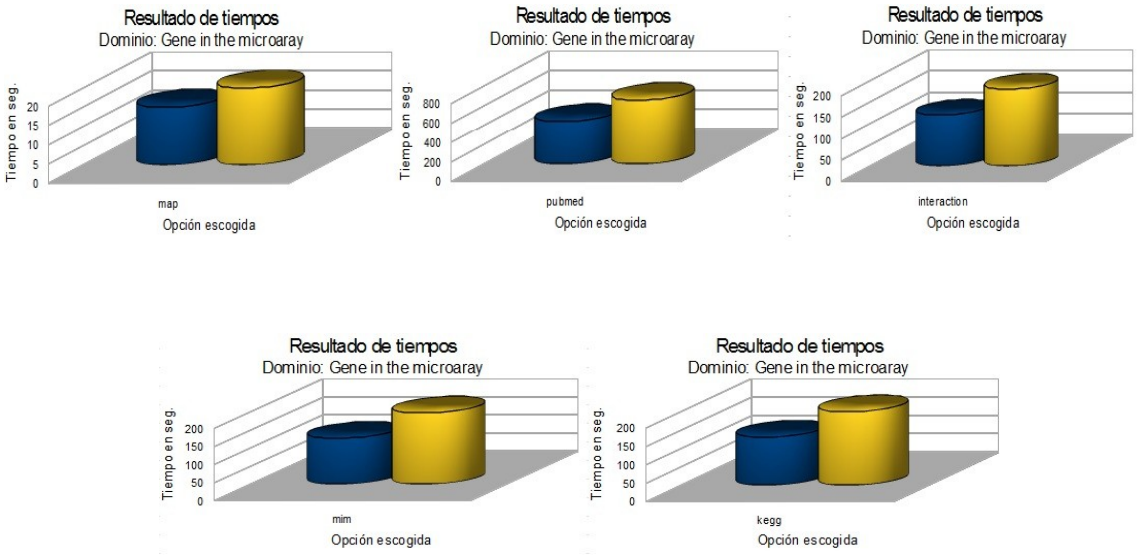
En este caso, estamos trabajando en el dominio de “Same Cluster”. Por lo tanto, sólo mostraremos genes que estén en el mismo *clúster* que los **genes de entrada**.

A continuación, mostraremos una tabla con los porcentajes de mejora:

Op. escogida	% mejora
map	35,28%
interaction	58,42%
pubmed	93,10%
kegg	83,58%
mim	86,69%
go function	79,96%
go process	84,95%
go component	84,97%

Tabla 3.1.2.3.2.4: Porcentajes de mejora "Same cluster"





Img. 3.1.2.3.2.3: Resultados tiempos "Genes in the microarray"

En esta muestra, hemos escogido el dominio de “Genes in the microarray”, por lo tanto, mostrará sólo los **genes relacionados** que estén dentro de la *microarray*.

Adjunto el porcentaje de mejora:

Op. escogida	% mejora
map	25,00%
interaction	33,72%
pubmed	33,20%
kegg	34,41%
mim	35,50%
go function	23,40%
go process	37,61%
go component	31,93%

Tabla 3.1.2.3.2.5: Porcentajes mejora "Genes in the microarray"

3.1.2.3.3. Reordenación de las relaciones génicas

Como ya he explicado anteriormente (en el apartado **3.1.1**), el aplicativo anterior permitía ordenar los resultados, una vez mostrados, recorriendo de nuevo los archivos que había generado volviendo a mostrar de nuevo los resultados. Esto ralentizaba mucho la ordenación.

Me encuentro en la situación de tener que ordenar los datos de una tabla que estoy mostrando en el aplicativo php, es decir, en código **html**, ya que no dispongo de los ficheros

que se generaban en el aplicativo anterior.

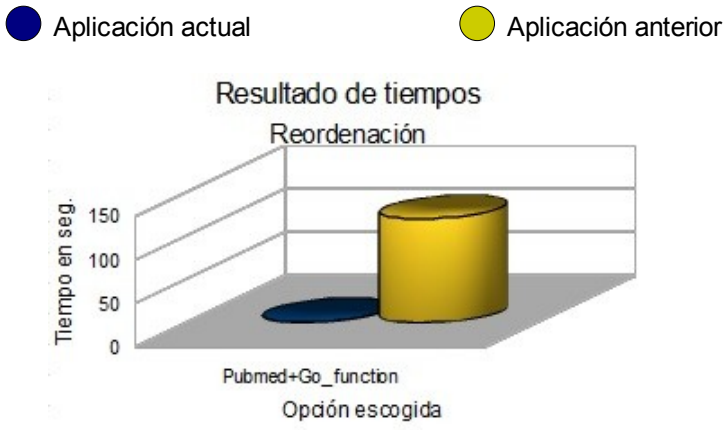
El usuario tiene que poder ordenar las **relaciones génicas** mostradas en el aplicativo php por el campo que deseé. Hasta el momento no era posible ordenar por nombre de gen.

Valoré dos opciones:

- Ordenar la información de las tablas auxiliares y volver a mostrar los datos referentes a la sesión del usuario una vez ordenados por el campo deseado.
- Ordenar la tabla que estoy mostrando directamente en la aplicación php, ya que es donde tengo los resultados ya cargados y no tenemos que volver a consultar a la base de datos.

Efectivamente, la mejor solución es la segunda, ya que de esta forma estamos ordenando los resultados que ya tenemos mostrados. De esta forma, no tenemos que volver a cargar la página. Para ello, inserté una nueva funcionalidad en java script, **tablesorter**[4], que ordenaba la tabla mostrada en html, de la misma forma que podemos hacer una consulta a la base de datos y ordenar por cualquier campo.

Este tipo de ordenación es más rápido que el que estaba implementado anteriormente:



Img. 3.1.2.3.3.1: Tiempo de ordenación aplicativo web

Podemos observar en el gráfico anterior, que la ordenación que he conseguido es inmediata. Mientras que la anterior, vuelve cargar los datos que tiene los ficheros *.xml* y *.txt*, y muestra de nuevo la tabla. El porcentaje de mejora de tiempo es del 100%.

Existe un gran inconveniente en la ordenación que hemos añadido en el proceso de mostrado de información del aplicativo php. El sistema de ordenación anterior volvía a cargar toda la información por un motivo, al ordenar por un campo el resto de información, se

4 http://tablesorter.com/docs/

mostraba en su totalidad por cada entrada del campo que estamos ordenando. Es decir, imaginemos que estamos haciendo una consulta y escogemos que nos muestre las **relaciones génicas** a unos **genes de entrada** de **Pubmed** y de **Kegg**.

En esta primera tabla encontramos los resultados ordenados por **Pubmed**.

Ordenación por Pubmed		
Pubmed	**Kegg**	**Genes**
Pub1	**Kegg1**	**Gen1**
	Kegg2	**Gen2**
Pub2	**Kegg1**	**Gen3**
Pub3	**Kegg2**	**Gen4**

Tabla 3.1.2.3.3.1: Muestra de diferentes ordenaciones

En el aplicativo anterior, si el usuario decidía ordenar por kegg, los resultados se mostraban de la siguiente manera:

Ordenación por Kegg		
Kegg	**Pubmed**	**Genes**
Kegg1	**Pub1**	**Gen1**
	Pub2	**Gen3**
	Pub3	**Gen4**
Kegg2	**Pub1**	**Gen1**
	Pub2	**Gen3**
	Pub3	**Gen4**

Tabla 3.1.2.3.3.2: Muestra de diferentes ordenaciones aplicativo anterior

De esta forma, a parte de estar ordenado alfabéticamente, sabemos en todo momento qué genes son comunes a un Kegg o a un Pubmed.

Con el método de ordenación que he insertado, mostraría los resultados de la siguiente manera:

Ordenación por Kegg		
Pubmed	**Kegg**	**Genes**
Pub1	**Kegg1**	**Gen1**
Pub2	**Kegg1**	**Gen3**
Pub1	**Kegg2**	**Gen2**
Pub3	**Kegg2**	**Gen4**

Tabla 3.1.2.3.3.3: Muestra de diferentes ordenaciones aplicativo actual

Al ordenar por kegg, no vemos todos los **Pubmeds** resultados por cada **Kegg.** Aunque no tengan relación **Kegg** y **Pubmed**, más que la que pueden tener a partir de un gen, tenemos que mostrar la información de tal forma que busquemos la relación entre ellos para hacerla legible al usuario.

Para solucionar este inconveniente, hemos decidido implementar unas nuevas funcionalidades, explicadas en la siguiente sección, que accederán de forma *on-line* a las tablas auxiliares y aumentarán considerablemente la potencia del aplicativo php respecto del anterior.

3.1.2.3.3.1. Nuevas funcionalidades de operaciones php

Una de las nuevas funcionalidades que queremos disponer es la búsqueda de **genes relacionados** a partir de unos **genes de entrada** que el usuario puede introducir manualmente. Este hecho implica habilitar una nueva opción en el applet *PCOPGene-net*, que explicaré más adelante en la sección **3.3.1**, además que la aplicación php recoja los diferentes nombres de los genes que ha introducido el usuario, y implementar un algoritmo de búsqueda que encuentre a que gen está relacionado estas nomenclaturas. Esta acción se ha conseguido gracias a que tenemos la información referente a las nomenclaturas de cada gen en la base de datos (en la estructura de la base de datos vemos un campo en las tablas gene_info_X que se llama Alias, donde guardamos todas las nomenclaturas de un gen) y utilizamos dichas nomenclaturas para discriminar los **genes de entrada**.

Otra funcionalidad muy importante, es la de habilitar la búsqueda de **relaciones génicas** sin necesidad de partir de ningún **gen de entrada**, ya sea seleccionado del *gene network* o introducido manualmente. Para esto, tenemos habilitado en el applet *PCOPGene-net* (sección 3.3.2) la posibilidad de introducir palabras clave en las opciones de Pubmed y Go function, process y component. La aplicación php es la encargada de realizar la búsqueda de

genes relacionados en la base de datos, discriminando dicha búsqueda por estas palabras clave. La razón es que tenemos los títulos en gene2go sobre los que se puede hacer búsquedas y, en la nueva tabla de PubMedInfo, guardamos los títulos de los *pubmeds* para hacer las consultas que necesitamos.

En la siguiente imagen podemos observar un *checkbox* con una descripción.

Gene	Protein_interaction	Reference	GO_Function	Reference
PARP1: poly (ADP-ribose) polymerase family, member 1 3 entry	Affinity Capture-Western; in vitro 1 Genes found	Ref. Ref.	protein N-terminus binding 3 Genes found	Ref.
PARP1: poly (ADP-ribose) polymerase family, member 1 3 entry	- 1 Genes found	Ref.	~~	~~
ATM: ataxia telangiectasia mutated 6 entry	in vivo; Phenotypic Enhancement; Protein-peptide; Reconstituted Complex 29 Genes found	Ref. Ref. Ref. Ref. Ref.	protein N-terminus binding 9 Genes found	Ref.
ATM: ataxia telangiectasia mutated 6 entry	- 1 Genes found	Ref.	~~	~~
ATM: ataxia telangiectasia mutated 6 entry	An unspecified isoform of ATM interacts with and phosphorylates p53. This interaction was modeled on a demonstrated interaction between human ATM and p53 from an unspecified species. 29 Genes found	Ref.	~~	~~

Img. 3.1.2.3.3.1.1: Opción de cruce on-line

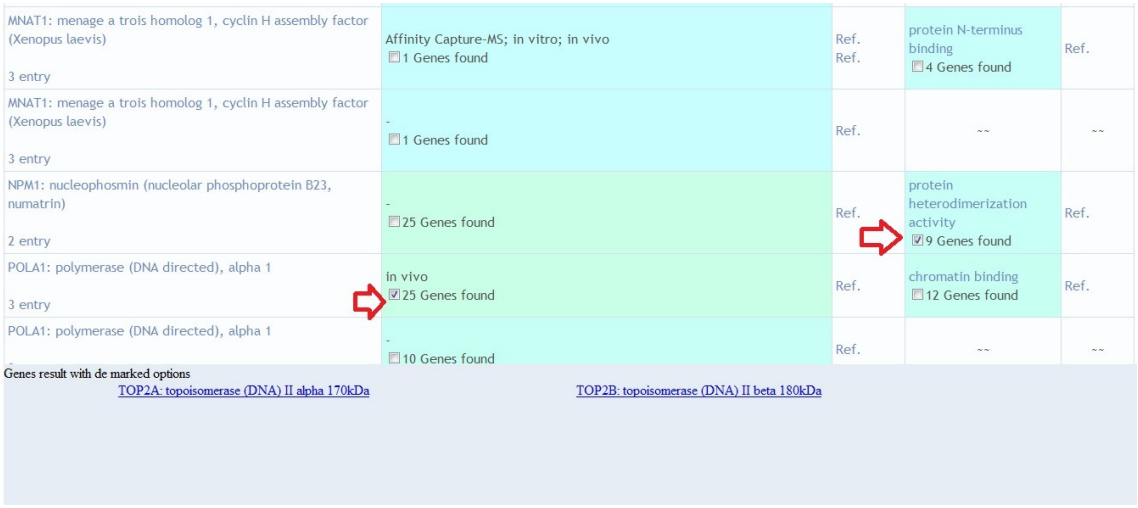
Todos las **relaciones génicas** entre los **genes de entrada** y los **genes relacionados**, que estamos mostrando en el aplicativo php, tienen esta opción.

Si leemos con atención la descripción del *checkbox*, vemos que tenemos un número. El número indica la cantidad de entradas -y con entradas me refiero a la suma del total de genes que tienen esta **relación génica**- para el total de **genes relacionados**.

Otra de las funcionalidades de este número es determinar el color de la celda que lo contiene, es decir, cuanto más alto sea el valor de este número, la celda tendrá un color más vistoso. Esta funcionalidad puede ser muy útil para determinar qué **genes relacionados** están más estudiados, dentro de la selección que hemos hecho en un principio.

Si marcamos el *checkbox* en la parte inferior de la pantalla, nos mostrará una nueva división con todos los genes comunes al *pubmed* seleccionado mediante *AJAX* y un *script php*, que lanzará una consulta a las tablas auxiliares. Por ejemplo, el usuario ha realizado la consulta con varios filtros -por ejemplo selecciona **Pubmed** y **Kegg**- y marcamos cualquier checkbox de cualquier **Kegg** y otro de **Pubmed**. En la nueva división nos va a mostrar los **genes relacionados** comunes a las **relaciones génicas** marcadas por el usuario. De esta forma podemos saber en todo momento, los genes comunes a un **Pubmed** concreto, entre los resultados mostrados. De esta manera, es una forma muy útil para ir acotando la información de las relaciones génicas y facilitar la investigación. Esta nueva división se mantiene al final de la pantalla (mediante diseño en CSS[17], idioma de maquetación de webs) es decir que,

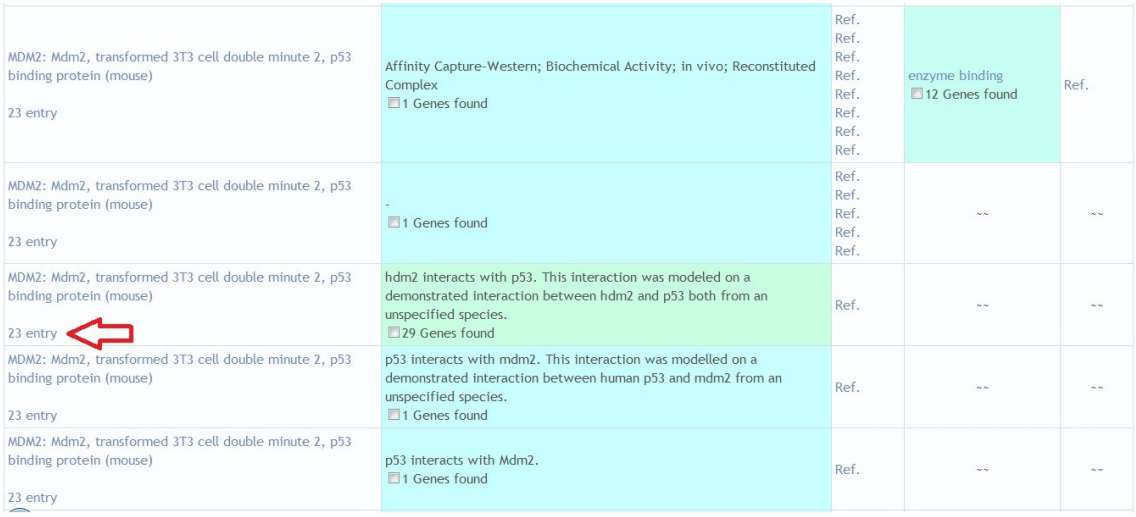
aunque naveguemos por toda la pantalla, siempre tendremos visibles los **genes relacionados** de las opciones que vayamos marcando.



Img. 3.1.2.3.3.1.2: Genes comunes a unos resultados marcados

En la imagen anterior, podemos observar cómo hemos seleccionado dos **relaciones génicas**, una que contienen 25 **genes relacionados** de entre todos los **genes relacionados** y otra que contiene 9 genes de entre todos los **genes relacionados**. Al marcar las dos opciones, se abre la división, que observamos en la parte inferior, con los genes comunes entre estos 25 y 9 de cada opción (en este caso solamente tenemos dos genes comunes de entre los 25 comunes a la primera opción y los 9 comunes a la segunda opción). Con esta opción podemos acotar más la información, para centralizar el estudio en unos genes en concreto.

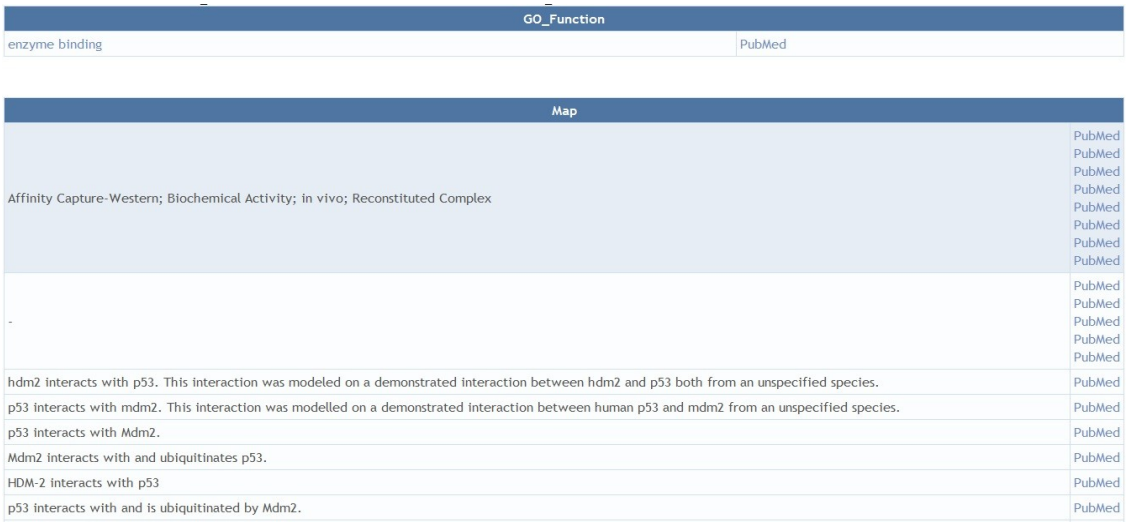
Otra nueva funcionalidad a este aplicativo es la que mostramos seguidamente:



Img. 3.1.2.3.3.1.3: Opción de mostrar información de un ***gen relacionado***

El número nos está indicando la cantidad de resultados asociados a este gen del total de resultados comunes a los **genes de entrada**. Como al ordenar perdemos, visualmente hablando, todas las **relaciones génicas** referentes a un **gen relacionado** respecto del total de resultados de la consulta. De esta forma podemos ver todos los datos referentes al gen seleccionado.

Al seleccionar esta opción se abrirá una ventana nueva con toda la información del gen seleccionado, dentro del conjunto de la información de las **relaciones génicas** comunes a los **genes de entrada**, es decir, que está opción también accede a las tablas auxiliares para seleccionar las **relaciones génicas** comunes a la sesión actual del usuario sobre el gen que está investigando.



GO_Function	
enzyme binding	PubMed

Map	
Affinity Capture-Western; Biochemical Activity; in vivo; Reconstituted Complex	PubMed PubMed PubMed PubMed PubMed PubMed PubMed PubMed
-	PubMed PubMed PubMed PubMed PubMed
hdm2 interacts with p53. This interaction was modeled on a demonstrated interaction between hdm2 and p53 both from an unspecified species.	PubMed
p53 interacts with mdm2. This interaction was modelled on a demonstrated interaction between human p53 and mdm2 from an unspecified species.	PubMed
p53 interacts with Mdm2.	PubMed
Mdm2 interacts with and ubiquitinates p53.	PubMed
HDM-2 interacts with p53	PubMed
p53 interacts with and is ubiquitinated by Mdm2.	PubMed

*Img. 3.1.2.3.3.1.4: Información de las **relaciones génicas** de un **gen relacionado***

Estas nuevas funcionalidades no existían en el aplicativo anterior y hacen que la potencia de consulta de esta aplicación crezca considerablemente. Con las dos primeras mejoras explicadas, podemos empezar el estudio de varias formas diferentes (a partir de las diferentes nomenclaturas de los genes o a partir de palabras clave que introduzca el usuario). Con las otras dos mejoras tenemos accesibles en todo momento las **relaciones génicas** que necesitemos estudiar, de una forma más acotada para facilitar el estudio de un grupo de **genes relacionados**, respecto de unos **genes de entrada**.

3.2. Actualización de la base de datos

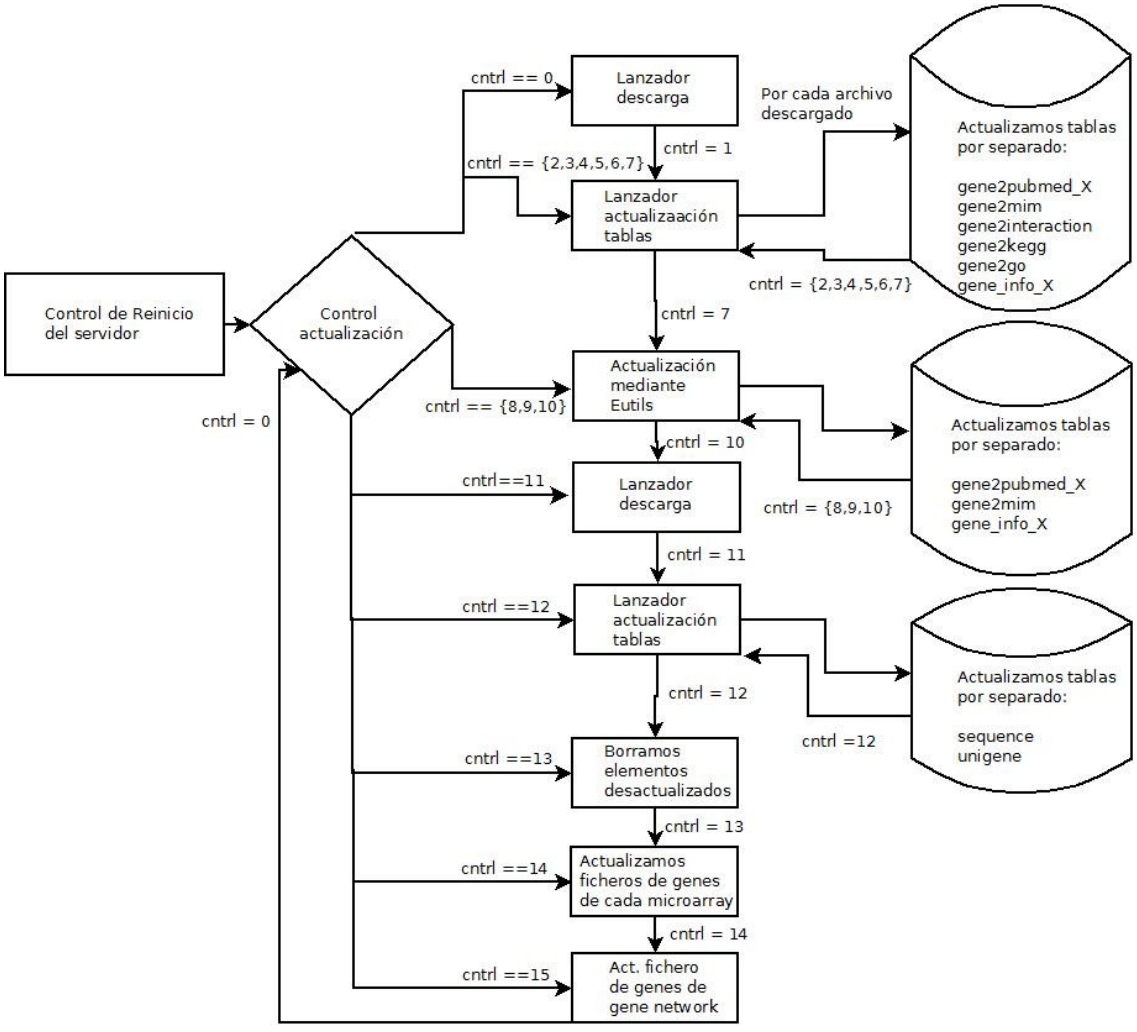
En esta sección, explicaré las dificultades que he encontrado en el momento de hacer la actualización, cómo los he solucionado y cómo la he adaptado a la nueva estructura de la base de datos.

En un primer momento, me encontré con unos archivos realizados en perl[18]. Perl[5] es un idioma de programación con el que resulta muy fácil el tratamiento tanto de archivos de texto como de archivos *.xml*. Más adelante, explicaré porque necesitaremos tratar estos tipos de archivos.

Estos archivos lanzaban la actualización de la base de datos local. La actualización no era de toda la base de datos, faltaba realizar la descarga de los archivos de actualización de la tabla *Kegg* conectándonos a la *ftp* de Japón, y la tabla *Map* desde el *ftp* de NCBI.

El *cron*[6] *daemon*[7] para programar tareas en los sistemas basado en *Unix*, no se encontraba programado para la actualización periódica y faltaba adaptar la actualización a la nueva estructura de la base de datos local.

La actualización de la base de datos que actualmente está accesible en el IBB, tiene el siguiente diagrama de flujo:



Img. 3.2.1: Diagrama de flujo de actualización

5 http://perlenespanol.com (foro de expertos en Perl)

6 http://es.wikipedia.org/wiki/Cron_%28Unix%29 (concepto de cron)

7 http://es.wikipedia.org/wiki/Demonio_%28inform%C3%A1tica%29 (definición de daemon)

3.2.1. Precondiciones

La programación de la actualización de la base de datos tiene que tener en cuenta otras actualizaciones que se realizan en diversas aplicaciones, para ello, evitaremos que coincidan en el mismo momento todas las actualizaciones.

La actualización de la base de datos es casi secuencial. Cuando acabamos una fase empieza la siguiente. Consideramos que es lo correcto, ya que de esta forma no colapsamos el servidor ni la base de datos, porque uno de los requisitos indispensables que se me exigieron fue que la base de datos tiene que estar siempre activa. Esto quiere decir, que la aplicación php está plenamente operativa mientras la base de datos local está actualizándose.

La actualización la realizamos sobre la base de datos original y no sobre un *backup*, los motivos de esta decisión son:

- Supongamos que hiciéramos la actualización sobre un *backup* de la base de datos original. Esto supondría tener que eliminar la base de datos original, cuando haya acabado la actualización, y renombrar el *backup*. Con este proceso, estamos dejando la aplicación inaccesible durante el tiempo que tarde en renombrar el *backup*.
- Tener que eliminar una base de datos entera, es muy ineficiente en cuestión de tiempo, dejaríamos el aplicativo inoperativo durante un período muy largo de tiempo.
- La base de datos tiene que estar siempre activa.
- En un plano de menor importancia, tendríamos que tener la base de datos duplicada, con el coste de espacio en disco que esto conlleva.

Con estos motivos, debía de implantar un sistema que permitiera tener la base de datos siempre accesible. Si nos fijamos en la estructura de la base de datos, podemos ver que todas las tablas que son sujeto de la actualización tienen un campo que se llama **Exist.** Este campo tiene un comportamiento similar a un *flag*, si es 0 se tiene que borrar este dato y si es diferente de 0 es que está desactualizado. Mediante este campo, controlaré qué datos son los que acabamos de insertar en la base de datos o han sido objeto de alguna modificación y qué registros no se han modificado y, por lo tanto, tienen que ser eliminados de la base de datos, ya que no existen en el archivo origen de la actualización.

Otro de los requisitos hace referencia a la descarga de los archivos de texto del *ftp* correspondiente. Los archivos que se encargaban de lanzar los robots de actualización de cada tabla y de descargar los archivos necesarios, realizaban la descarga del archivo y seguidamente actualizaban la tabla correspondiente. Consideramos que es incorrecto este flujo de ejecución ya que podríamos generar inconsistencias en la base de datos. Por ejemplo, imaginemos que estamos descargando el archivo referente a la actualización de las tablas de **gene_info_X**. Esta tabla es de las más grandes de la base de datos, como ya hemos

comentado anteriormente. Esto implica que hasta que no acabe de actualizar las tablas de **gene_info_X**, no empezará la descarga de otro archivo. Seguidamente a la actualización de la tabla **gene_info_X**, empezará la descarga de cualquier otro archivo, con la posibilidad de que éste haya sido modificado, añadiendo alguna entrada con algún *GeneID* que no se encuentre en las tablas de **gene_info_X**. De esta manera, tendríamos datos que nunca íbamos a acceder en la base de datos, puesto que no tenemos el GeneID en las tablas **gene_info_X**.

Con este requisito decidimos adaptar la actualización, para que realizara la descarga de los archivos todos juntos, y una vez descargados, empezará la actualización de la base de datos.

Para ello, si observamos la base de datos, podemos ver dos tablas llamadas **gene_ftp** y **unigene_ftp**. En estas tablas guardamos el nombre de los archivos que nos descargamos, junto con el tamaño que ocupan, y la fecha de la última modificación en el servidor del *ftp* que lo estamos descargando. Esto nos sirve para controlar la actualización en un momento de corte de servicio, por ejemplo si se ha reiniciado nuestro servidor mientras estaba realizando la actualización.

El control de la actualización lo realizamos mediante el *cron*. En el momento de que sucede un reinicio del servidor, el *cron* captura el evento cuando el sistema esta levantado. Mediante un *log* que vamos escribiendo en unos archivos vamos controlando en que fase de la actualización nos hemos quedado en el momento del corte del servicio y seguiremos la actualización desde ese momento. Imaginemos que nos hemos quedado en la descarga de los archivos, las tablas **gene_ftp** y **unigene_ftp** nos dicen que archivos están descargados correctamente y que archivos tenemos que volver a descargar.

3.2.2. Actualización robusta adaptada a la nueva estructura de la BBDD

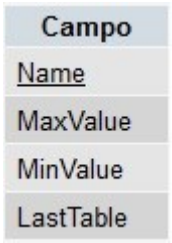
3.2.2.1. Tablas divididas

Ya tenemos la parte del aplicativo php adaptado a la nueva estructura de la base de datos que hemos implementado. Ahora tenemos que implementar estas modificaciones en la fase de actualización. El aplicativo php sólo necesitaba adaptarse a la base de datos en el momento de acceder a los datos -ya hemos explicado que lo hacemos mediante la tabla TableName (sección 3.1.2.3.1)- donde guardamos todos lo nombres de las tablas existentes de gene_info_X y de gene2pubmed_X.

En esta fase -la fase de actualización- tengo que adaptar la estructura de la base de datos en el momento de acceder a los datos que tenemos en las diferentes tablas, para hacer inserciones o modificaciones. También debo de implementar un algoritmo que se adapte a la inserción masiva de datos, con esto me refiero, a que la estructura de la base de datos tiene que ir adaptándose a la gran cantidad de información que será insertada y mantener una

cantidad de registros/tabla razonable. Las tablas sujetas a estas modificaciones, como bien se intuye, serán gene_info_X y de gene2pubmed_X.

Si nos fijamos en la estructura de la tabla **TableName**, podemos observar lo siguiente:



Img. 3.2.2.1.1: Campo de TableName

En el momento en que lanzamos los archivos que actualizan las tablas antes mencionadas, estos irán leyendo linea por linea, y por cada *GeneID* o por cada PMID que lean, recorrerán las tablas contenidas en *TableName*, dependiendo si son *GeneID* o PMID recorrerán gene_info_X y de gene2pubmed_X respectivamente. En el momento en que los valores de GeneID o PMID estén dentro de los intervalos de *MaxValue* y *MinValue*, cogeremos el *Name* correspondiente a la tabla donde tenemos que insertar la información de la linea del archivo que estamos leyendo actualmente, y la insertaremos.

Antes de insertar la información, miraremos si ya existe dicho identificador en la tabla que estamos consultando. Si ya existe, nos limitaremos a modificar los datos que sean modificables, todos menos el campo clave que es el identificador. Si no existe en la tabla, el dato que estamos consultando, insertaremos el nuevo dato en la tabla correspondiente. Este proceso es similar al que realizábamos en el aplicativo php para seleccionar los genes o los Pubmeds mediante el campo identificador.

En el archivo de actualización de gene_info_X y de gene2pubmed_X, se tienen que poder crear tablas automáticamente dependiendo de unos criterios para controlar el tamaño de cada tabla. Si observamos la estructura de *TableName* vemos un campo que se llama *LastTable*. Este campo actuará de flag para indicar cual es la última tabla que se ha construido.

La necesidad de crear una nueva tabla viene dada por el identificador que se esté leyendo en ese momento. Si el identificador es más grande que el *MaxValue* de la tabla que tenga en *LastTable* a 1, es decir la última tabla creada, entonces se creará una nueva tabla con el nombre de la última tabla creada gene_info_X+1 o gene2pubmed_X+1. El valor de *MinValue* de esta nueva tabla que se ha creado viene dado por el *MaxValue* de la tabla anterior, y el *MaxValue* de la nueva tabla viene dado, en el caso de gene_info_X, por el *MaxValue* de la tabla anterior más 300.000. Y en el caso de gene2pubmed_X será más 500.000.

He escogido estos valores después de realizar un estudio de las tablas que tenía divididas en la base de datos con un número más o menos equivalente de elementos. En cada caso, el valor que estoy sumando al mínimo de cada tabla es la media de la diferencia entre los *MaxValue* y *MinValue* de cada una de las tablas que ya están creadas. La nueva tabla que hemos creado pasará a tener el *flag* de *LastTable* activado, mientras que en la tabla que lo tenia anteriormente lo desactivaremos.

En la actualización de las tablas de gene2pubmed_X hay que tener en cuenta que también tenemos que actualizar PubmedInfo, Es decir, si modificamos un Pubmed, sólo modificamos el registro que esté en PubmedInfo dejando el título a NULL para después actualizarlo. No modificamos la tabla de gene2pubmed_X correspondiente a ese PMID porque los genes que contenía esa publicación no se ven afectados, siguen siendo los mismos. Si insertamos un dato, insertamos tanto en la tabla gene2pubmed_X correspondiente, como en la tabla de PubmedInfo.

De esta forma controlamos que se vayan creando nuevas tablas mientras lo vayamos necesitando, con unos criterios determinados.

3.2.2.2. Actualización mediante *Eutils*

Como ya hemos comentado, la actualización de la base de datos está realizada en *perl* ya que nos facilita mucho el tratamiento de archivos tanto de texto como *.xml.* En el caso de los archivos .xml, he instalado un modulo en el servidor del IBB que necesitaba para tratar los *.xml.* Estos módulos están de forma gratuita en la página de CPAN[19].

Una vez adaptadas las actualizaciones a las tablas divididas, tenemos que afrontar la actualización de las tablas de gene_info_X, PubmedInfo y gene2mim, mediante archivos *.xml* descargados del *webservice* del NCBI mediante las herramientas *Eutils*.

Las consultas mediante las Eutils no son ilimitadas, tienen restricciones:

- No sobrecargar el sistema del NCBI. Los usuarios que intenten enviar numerosas peticiones o consultar resultados demasiado complejos, se les prohibirá el acceso durante un periodo de tiempo.
- Ejecutar consultas los fines de semana o durante horas de madrugada (hora estadounidense).
- Enviar las consultas a http://eutils.ncbi.nlm.nih.gov en lugar de a la dirección estándar del NCBI.
- No hacer más de una petición cada 3 segundos.

Para esta sección de la actualización haremos uso del *flag* **Exist**, que tienen todas las tablas sujetas a la actualización.

El *flag* de todos los elementos que han sido actualizados es diferente de 0, por lo tanto, en las tablas que vamos a actualizar (gene_info_X, gene2pubmed y gene2mim), iremos seleccionando los elementos con el *flag* **Exist** igual a 1, de 500 en 500, es decir, en cada consulta, mediante las *Eutils***,** enviaremos 500 elementos de los que queremos conocer las nomenclaturas, en el caso de **gene_info_X** o los títulos en el caso de **gene2pubmed** y **gene2mim**. He elegido seleccionar elementos de 500 en 500, para evitar que el servicio de *webservice* cierre nuestra conexión por cierto tiempo. Más de 500 elementos es excesivo para el NCBI y menos elementos, ralentizaría mucho la actualización.

Después de actualizar los datos mediante las consultas al *webservice* del NCBI, cambiaremos el *flag* **Exist** para tener controlados los elementos que hemos actualizado y los que no.

Al terminar la actualización de las tablas mediante las consultas *Eutils*, puede ser que nos encontremos con datos que tienen el *flag* **Exist** a 0. Estos tienen que ser borrados, ya que son elementos desactualizados. Este proceso va seguido después de cada actualización de las tablas **gene_info_X, gene2pubmed** y **gene2mim**.

3.2.2.3. Borrado de elementos desactualizados

Cuando acaba este proceso de borrado, lanzamos un *daemon* que se encargará de borrar todos los elemento de las demás tablas que sigan teniendo el *flag* a 0.

Seguidamente, al acabar el borrado de los elementos desactualizados, procedemos dejar la base de datos como estaba en un principio, con el *flag* a 0 de todas las tablas, para la actualización siguiente dentro de tres meses.

3.2.3. Generación del archivo actualizado para creación del grafo

A continuación, otro de los requisitos de la actualización era la generación de un archivo con los nombres correctos de los genes, una vez actualizados.

Para ello tenemos dos *daemons* que son los encargados de actualizar los archivos que contienen los genes de las *microarrays*.

- Se recorrerán los directorios locales de *microarrays*, buscando en ellos los ficheros .genesorig (contienen nombre de los genes de dicha *microarray*), sustituyendo cada

línea por el nombre de gen actualizado correspondiente al gen de dicha línea (formato de nombre de gen: "*Symbol: Nombre Del Gen"*).

- Los ficheros *.genesorig* de las *microarrays* existentes quedan actualizados correctamente con los nombres de gen en el fichero *.genes*. El fichero *.genesorig* queda tal cual, sin modificar.
- Cada *microarray* tiene un fichero *.genesorig*.

Como tenemos los nombres de los genes actualizados, accederemos a la tabla de **gene_info_X** correspondiente para seleccionar el nombre del gen que necesitamos. A continuación, actualizamos el archivo, que estemos modificando, y la tabla de genes que tenemos en la base de datos.

La tabla **genes** contiene los nombres actualizados de los genes de cada *microarray* existente en nuestro servidor.

A continuación, procedemos a actualizar el fichero que tiene que leer *PCOPGene* para generar el grafo de genes de la *microarray*, para ello basta con copiar los genes actualizados del archivo 17.genes en el archivo destino. Sólo necesitamos los genes de la *microarray* 17, que es la que carga el *applet* de java.

3.3. Modificación del applet PCOPGene-net

Aquí presento la última fase del trabajo, que consiste en la modificación de applet *PCOPGene-net* para añadir las nuevas funcionalidades planteadas en los objetivos. Estas modificaciones las he realizado sobre el ejecutable del proyecto directamente, importando este ejecutable comprimido en .jar[8](archivo comprimido y ejecutable) al Eclipse[20].

Las nuevas características consisten en añadir la opción de poder consultar los genes mediante sus nomenclaturas, mediante las palabras clave introducidas por el usuario y añadir la opción para que la aplicación marque los **genes relacionados** de la consulta que visualizamos en el aplicativo php, en el *gene network* de la *microarray*.

3.3.1. Búsqueda por nomenclatura de gen

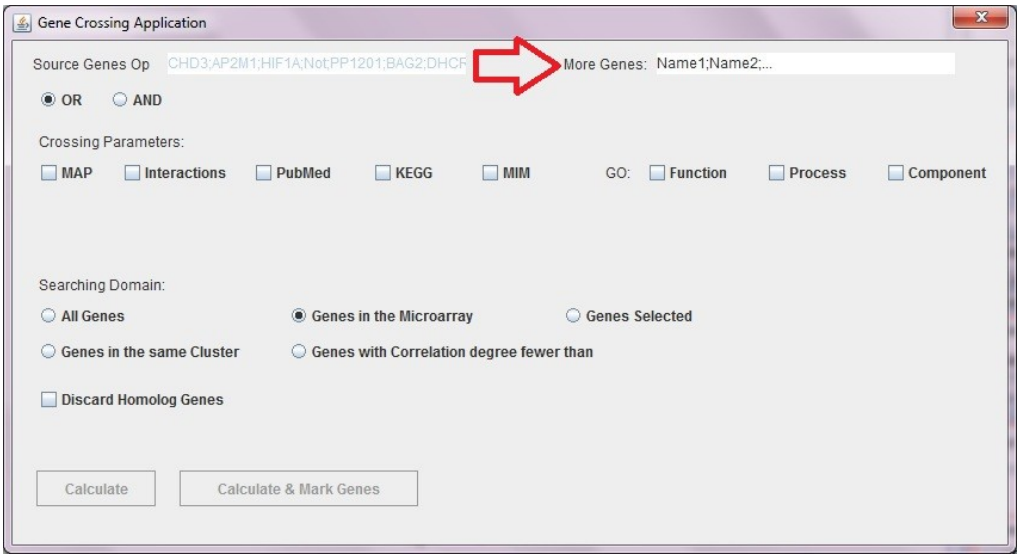
En un principio, teníamos previsto que *PCOPGene-net* seleccionase el gen correspondiente a las nomenclaturas que introduce por teclado el usuario.

8 http://jar.extensionfile.net/es

Luego, decidimos que no permitiríamos a *PCOPGene-net* acceder a la base de datos por temas de seguridad, ya que se ejecuta en local (en el ordenador del usuario) y no en el servidor, y habilitando el acceso del *PCOPGene-net* a la base de datos local abriríamos un agujero de seguridad importante.

Con este problema de seguridad, decidimos que el encargado de acceder a la base de datos seguiría siendo el aplicativo php. Esto implica que tenemos que enviar las nomenclaturas que introduce el usuario manualmente del *PCOPGene-net* al aplicativo php para que este busque el nombre del gen correspondiente.

Seguidamente, mostramos una imagen de las modificaciones de *PCOPGene-net* para permitir que el usuario pueda interactuar introduciendo las nomenclaturas de los genes que desee.

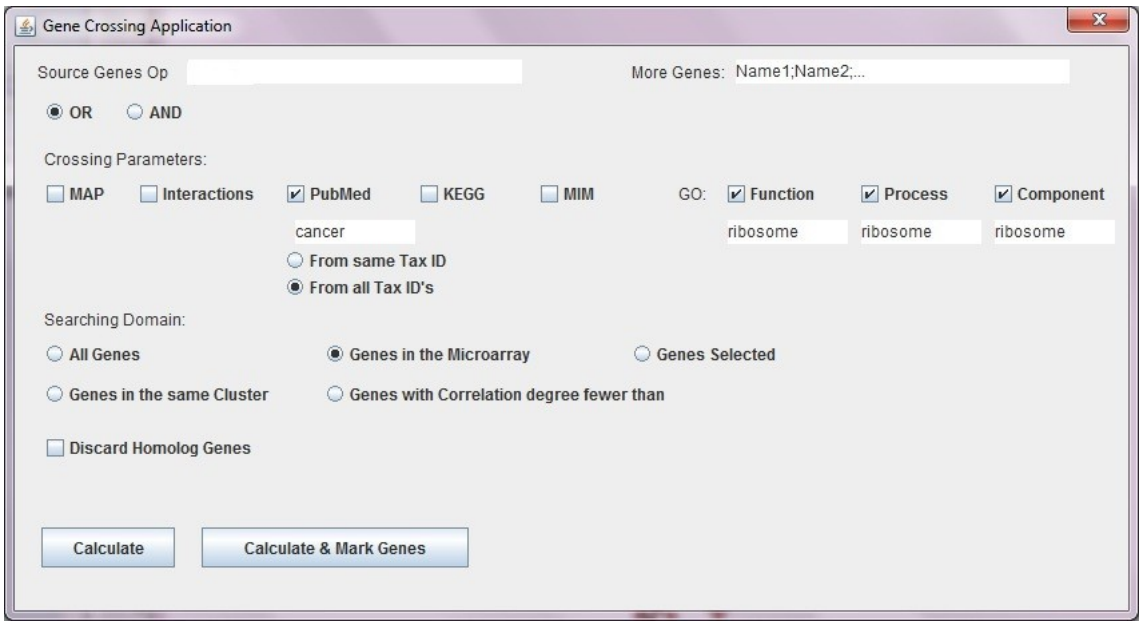


Img. 3.3.1.1: Opción de introducir nombre manualmente

3.3.2. Búsqueda sin inserción de genes de entrada

He habilitado la opción de poder buscar resultados sin tener que introducir genes de entrada manualmente, por teclado. O sin tener que marcar genes de entrada en el *gene network* original de la *microarray*.

Con esta nueva funcionalidad, podemos ver cualquier **gen relacionado**, en un filtrado por palabras clave en las tablas de **gene2pubmed** y **gene2go**, ya que éstos permiten introducir conceptos por los que podemos buscar.



Img. 3.3.2.1: Opción de consultar sin introducir ***genes de entrada***

La aplicación php es la encargada de mostrar los **genes relacionados** y las **relaciones génicas** comunes a una palabras clave insertadas por el usuario.

Este funcionalidad es muy importante, desde el punto de vista del investigador, ya que ofrece la posibilidad de buscar **relaciones génicas** independientemente de la entrada. De este modo, el usuario puede empezar investigar por otras vías y no necesariamente introduciendo unos **genes de entrada**, que acotan los caminos a seguir desde un principio (puede ser interesante acotar los resultados desde un principio o no, por eso habilito las dos opciones).

3.3.3. Habilitar el marcado de genes relacionados

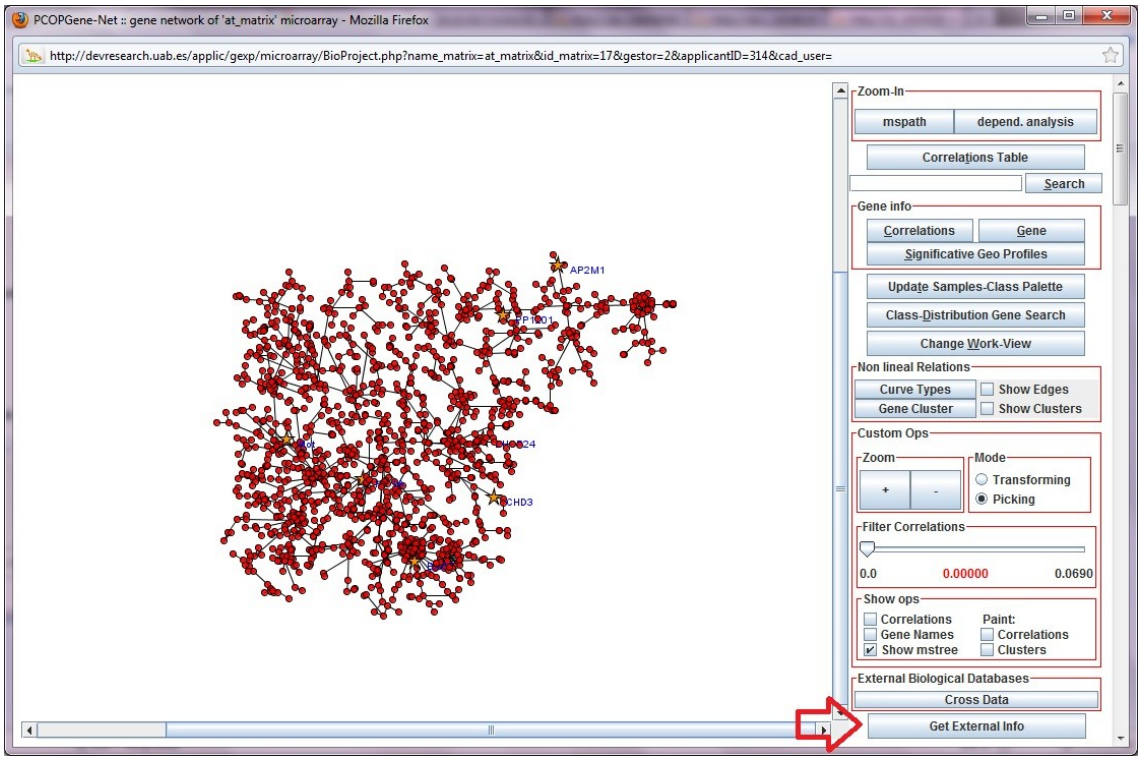
Para añadir la opción de marcado de los **genes relacionados** en el aplicativo *PCOPGene-net*, tenemos que habilitar esta funcionalidad, que ya está implementada, para que se active cuando ejecutamos la consulta hacia la aplicación php.

Para ello tenemos que generar el nombre del archivo, donde almacenaremos los **genes relacionados** que tiene que marcar, y pasarlo al aplicativo php.

En este archivo introducimos el identificador de la posición del gen en la *microarray*, almacenado en la tabla genes. Para ello necesitamos el identificador de la *microarray*, que lo enviamos del *PCOPGene-net* al aplicativo php encargado de operar con las **relaciones génicas**. Y el nombre del gen, como los **genes relacionados** los vamos imprimiendo linea a linea en la aplicación php al mismo tiempo que imprimimos el nombre del **gen relacionado**, consultamos a la tabla de genes e imprimimos el identificador del gen en la *microarray* en el

archivo de marcado.

Ahora el usuario puede ver los **genes relacionados** marcados en el grafo de la *microarray* sólo con presionar el botón indicado en la siguiente imagen.



*Img. 3.3.3.1: Opción de marcar **genes relacionados** en gene network*

En la imagen, podemos observar, que tenemos marcados los **genes relacionados** pertenecientes a la *microarray*, de la consulta que hemos lanzado.

4. Informe técnico

4.1. Aplicativo php

La aplicación php se encuentra en el siguiente path en cualquiera de los dos servidores del IBB: **/var/www/html/applic/gexp/microarray/cruces/mostrar.php**

Para visualizar resultados en este aplicativo php, construimos una url como la siguiente:

http://devresearch.uab.es/applic/gexp/microarray/cruces/mostrar.php?opc=2&time=18-14-23&tax=9606&gene=ETS2;&dom=1&microarray=17&umbral=0.1&topicfunc=&topiccomp=&topicproc=&use_tax=0&topicpub=&use_homo=0&logic=0&interactionOP=0&keggOP=0&pubmedOP=0&mimOP=1&mapOP=0&goFunctionOP=0&goComponentOP=0&goProcesOP=0&alias=lfs1;&nomFile=CDGS_1652011181423900

El encargado de construir esta url es el applet de java, que explicaré más adelante.

En esta url vemos todas las opciones que necesitamos en el aplicativo php, para realizar los cruces de información que desea el usuario:

url	
Opciones	**Descripción**
tax	Usada para determinar, junto **gene**, que gen está buscando el usuario.
gene	Se usa para determinar que **Source Gen** está buscando el usuario junto con **tax**.
dom	Identificamos que dominio ha escogido el usuario, para restringir la búsqueda.
microarray	Identificamos la microarray de los **genes de entrada**, para poder identificar y marcar los **genes relacionados** en el *gene network*.
umbral	Define que umbras tenemos de búsqueda de los **homologenes**.
topicfun, topiccomp, topicproc	Palabras clave que podemos usar para restringir la búsqueda en Go.
use_tax	Determina si el usuario quiere usar **tax** para la búsqueda o no.
topicpub	Permite introducir al usuario palabras clave para la búsqueda en **pubmed**.
use_homo	Indica si usamos homologene, para la búsqueda de genes.
logic	Indica que opción quiere hacer el usuario, un **ADN** o un **OR**. Para el cruce de información resultante.
interactionOP, keggOP pubmedOP, mimOP, mapOP, goFunctionOP, goComponentOP, goProcesOP	Identificador de la opción que quiere hacer el usuario.
alias	Opción por la que enviamos al php los alias que quiere consultar el usuario.
NomFile	Nombre del fichero en el que escribiremos, desde el php, los identificadores de los genes para marcarlos en el *gene network*.

Tabla 4.1.1: Opciones de la url

Mostrar.php es el encargado de evaluar todas las opciones que recibe, y mostrar los resultados en consecuencia.

Para la reordenación de la tabla que visualizamos en mostrar.php, he añadido una función java que ordena los datos de una tabla que ya está imprimida en html. Es una ordenación muy rápida e intuitiva.

Mediante los checkbox, asociados a cada uno de los resultados obtenidos, he implementado una función AJAX que captura si marcamos el checkbox o si lo desmarcamos. En tal caso, lanza una consulta a la base de datos, que es la encargada de encontrar todos los genes comunes a esa opción que hemos marcado. Si marcamos más de una opción, nos mostrará la unión de los genes comunes de las opciones que hemos marcado, en la misma página.

Junto con los checkbox, vemos un número que nos está indicando cuantos genes comunes, dentro de los **genes relacionados**, tiene asociado este resultado.

Debajo de los nombres de cada **gen relacionado**, vemos un link que nos llevará a una nueva página donde podemos ver toda la información referente a ese gen, dentro de los resultados obtenidos.

Junto a el link antes comentado, vemos un número que nos indica el número de entradas que tiene este gen, dentro de todos los resultados.

Mostrar.php, es el encargado de generar el archivo con los identificadores de los genes, para que se puedan marcar en el *gene network*. Este archivo se genera en el siguiente path del servidor:

/var/www/cgi-bin/pcop/microarray/classresults/"nomFile.txt"

El nombre del fichero "nomFile.txt", se construye en el applet de java que más adelante explicaré.

4.2. Actualización

La herramienta que utilizo para programar tareas en un sistema operativo basado en unix es **cron**, mediante los comandos:

- **Crontab -l:** visualizamos el contenido del cron.
- **Crontab -e:** editamos el cron con el editor por defecto en el servidor.

Introducimos en el cron la siguiente linea para programar la actualización trimestralmente:

0 0 1 */3 * bsh /var/www/cgi-bin/robots/lanzador.bsh

Las opciones del cron son las siguientes:

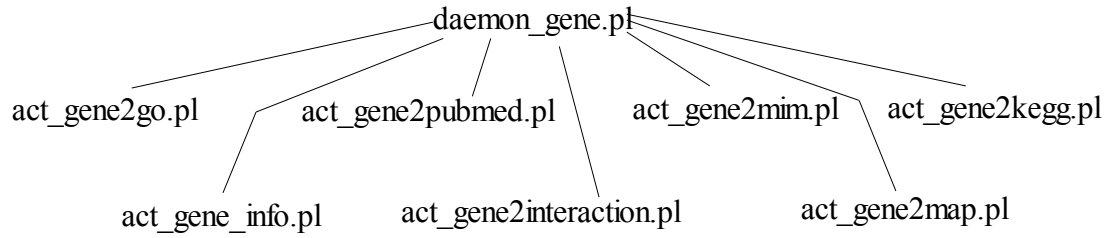
```
*  *  *  *  * comando o programa a ejecutar
|  |  |  |  |
|  |  |  |  |----- día de la semana (0 - 6) (0-> Domingo)
|  |  |  |-------- Mes (1 - 12)
|  |  |---------- Día del mes (1 - 31)
|  |------------- Hora (0 - 23)
|---------------- Minuto (0 - 59)
```

Img. 4.2.1: Configuración del cron

Ejecutamos el bash que lanza la actualización, porque los permisos del usuario propietario del cron no pueden ejecutar programas en perl directamente.

El path en el servidor de los daemons de la actualización es el siguiente: **/var/www/cgi-bin/robots/**

Lanzador.bsh es el encargado de ejecutar el primer paso de la actualización que es **daemon_gene.pl**.



Img. 4.2.2: Daemons lanzados por daemon_gene.pl

- **daemon_gene.pl:** Programa que actualiza las bases de datos locales de gene para obtener todas las características de los genes a partir de su GeneID.

  Para ello, crearemos una conexión FTP con el servidor remoto correspondiente. Nos bajaremos los ficheros que contienen información necesaria para realizar las consultas cruzadas sobre los genes. Estos ficheros tienen la siguiente ruta en el NCBI:

  - ftp.ncbi.nih.gov/gene/DATA/
  - ftp.ncbi.nih.gov/gene/GeneRIF/

  Y la correspondiente en kegg:

- ftp.genome.jp/pub/kegg/genes/

Antes de bajar los ficheros, se consultará en la tabla local gene_ftp la fecha y tamañano del último fichero que se bajó. Si coincide con el actual, no se volverá a bajar ya que los datos estarán actualizados. Si no coincide o no hay entrada en dicha tabla para ese fichero, se procederá a bajar y a insertar los datos en la BD.

Una vez insertados, se actualizará dicha tabla con la fecha y tamaño del fichero bajado.

Seguidamente llamará a los robots encargados de actualizar las tablas correspondientes.

- **act_gene2go.pl:** Programa que actualiza la base de datos con los valores de ontología para cada gen.

  Partiendo del fichero gene2go, extraemos sólo la informacion que nos será util en la BD local.

  Nos quedaremos con los campos: Tax_id, GeneID, GO_ID, Evidence, Qualifier, GO_term, PubMed y Category.

- **act_gene2interaction.pl:** Programa que actualiza la base de datos con los valores de interacciones entre genes.

  Partiendo del fichero geneinteraction, extraemos sólo la información que nos será útil en la BD local.

  Nos quedaremos con los campos: Tax_id, GeneID, interactantID, PubMed y GeneRif_text.

  El GeneID corresponde al primer interactuante y el codigo del segundo viene dado por interactantID.

  Si este codigo no es un gen (indicado por "-") no se añadirá la interacción a la base de datos.
  Pubmed es una lista con todos los artículos dónde sale la interacción.

- **act_gene2map.pl:** Programa que actualiza la base de datos con los valores de posición de un gen en el cromosoma.

  Partiendo del fichero gene2refseq, obtenemos la posición dónde empieza el gen a partir de su GeneID. Así podemos conocer los vecinos de dicho gen, aplicando un rango a ambos lados de esta posición dada.

Nos quedaremos con los campos: GeneID, GI y Start.

- **act_gene2mim.pl:** Programa que actualiza la base de datos con los valores de patologías para cada gen.

  Partiendo del fichero mim2gene, extraemos sólo la informacion que nos será útil en la BD local.
  Nos quedaremos con los campos: MIM_number, GeneID y type.

- **act_gene2pubmed.pl:** Programa que actualiza la base de datos con las publicaciones en las que aparece cada gen.

  Partiendo del fichero gene2pubmed, extraemos sólo la información que nos será útil en BD local.
  Nos quedaremos con los campos: Tax_id, GeneID, PMID.

- **act_gene_info.pl:** Programa que actualiza la base de datos con los valores de Gene para cada especie.

  Partiendo del fichero gene_info, extraemos sólo la información que nos será útil en la BD local.
  Nos quedaremos con los campos: Tax_id, GeneID, Symbol, Description y Alias.

- **act_gene2kegg.pl:** Programa que actualiza la base de datos con los valores de mapas del KEGG para cada gen.

  En este caso se parte del fichero KO obtenido del FTP remoto del KEGG. De este fichero extraemos la información relevante para cada gen.

  Nos quedaremos con los campos: Description, KO y Nombre del mapa del KEGG. Puesto que KEGG no es una base de datos del NCBI, no trabaja con GeneID's. En este caso, para hacer corresponder un gen con sus mapas del KEGG, lo hacemos a partir de su nombre completo.

Los programas de actualización de cada tabla, irán cambiando el flag de **Exist** de cada registro actualizado para tener un control de los registros que vamos actualizando y los que tenemos desactualizados.

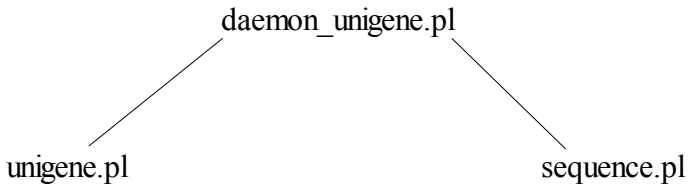
Cuando acabe la actualización de todas estas tablas, lanzaremos los tres robots que actualizan los datos mediante las herramientas Eutils que acceden al *webservice* del NCBI:

- **actPubMedInfo.pl:** conectaremos al *webservice* del NCBI mediante las herramientas Eutilscon la siguiente url:

http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=pubmed&id="id",
Del .xml que recibimos, seleccionaremos sólo los títulos de los pubmeds que tengamos en la tabla de **PubMedInfo**. Volveremos a actualizar el flag de **Exist** para tener controlados los registros que vamos actualizando el título, ya que no se pueden enviar consultas de muchos elementos mediante las Eutils porque nos limitarían el acceso durante un tiempo.

- **actGeneInfo.pl:** conectaremos vía *webservice* con el NCBI desde la siguiente url: http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=gene&id="id", para seleccionar del .xml los alias de los genes para actualizar la tabla gene_info_X. De la misma manera que en el anterior, actualizamos los flags por el mismo motivo.
- **actMIM.pl:** establecemos conexión con el NCBI, para descargar los títulos que tenemos que actualizar en gene2mim desde la url: http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=omim&id="id".

Al terminar con la actualización mediante las herramientas Eutils, lanzamos el **daemon_unigene.pl**.



Img. 4.2.3: Daemons lanzados por daemon_unigene.pl

- **daemon_unigene.pl:** Programa que actualiza las bases de datos locales de Unigene, para obtener los nombres de gen a partir de códigos de secuencia y de códigos Unigene.

  Para ello, crearemos una conexión FTP con el sitio UniGene del NCBI. Recorreremos todos los directorios de especies y de cada uno de ellos, tomaremos dos ficheros, el que contiene las secuencias asociada a dicha especie (xx.seq.all) y el que contiene la asociación de Codigo_UniGene con GeneID (xx.data).

  Antes de bajar los ficheros, se consultará en la tabla local unigene_ftp la fecha y tamaño del último fichero que se bajó. Si coincide con el actual, no se volverá a bajar ya que los datos estarán actualizados. Si no coincide o no hay entrada en dicha tabla para ese fichero, se procederá a bajar y a insertar los datos en la BD.

  Una vez insertados, se actualizará dicha tabla con la fecha y tamaño del fichero bajado.

- **Sequence.pl:** Programa que actualiza la base de datos con las secuencias de UniGene para cada especie.

  Para ello, partimos de los ficheros con formato xx.seq.all (xx es la especie) bajados del sitio ftp de Unigene. De este fichero tomamos, para cada identificador Unigene todos los códigos de secuencia que se le pueden asociar.

  El formato de este fichero .seq.all puede verse en
  ftp.ncbi.nih.gov/repository/UniGene

  Ejemplo de formato de línea del fichero para cada gen de la especie:

  ... /clone_end=3' /gb=AA985088 /gi=3163613 /ug=Hs.632256 / ...

  Nos quedaremos con el campo GB y UG. El código GB indica la secuencia y el UG el código de UniGene.

  Este programa perl pues, a partir de los ficheros .seq.all bajados de cada especie, actualiza los datos de la tabla local **sequence**.
- **Unigene.pl:** Programa que actualiza la base de datos con los valores de UniGene para cada especie.

  Para ello, partimos de los ficheros con formato xx.data (xx es la especie) bajados del sitio ftp de Unigene. De este fichero tomamos, para cada identificador Unigene el código de Gene asociado (si hay).

  El formato de este fichero .data puede verse en
  ftp.ncbi.nih.gov/repository/UniGene

  Nos quedaremos con el campo ID, GENE_ID.

  De esta forma, actualizamos los datos que asocian un código UniGene con un GeneID, pudiendo así trabajar a partir de entonces con códigos GeneID's.

  Este programa actualiza la tabla **unigene.**

Seguidamente, lanzaremos el daemon encargado de borrar todos los datos que no se encuentran actualizados.

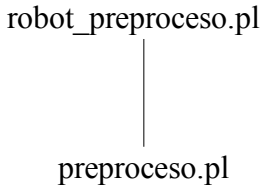
- **RemoveElements.pl:** Es el encargado de borrar todos los elementos de todas las tablas que tengan el flag de **Exist** igual a 0.
  Cuando termina de borrar todos estos elementos, borrar los archivos que nos hemos

descargado de los ftp correspondientes y que se encuentran en:

- **/var/www/cgi-bin/robots/temp_unigene/**
- **/var/www/cgi-bin/robots/temp_gene/**

Y vuelve a dejar el flag de todos los elementos a 0, para la siguiente actualización.

A continuación, lanzamos los daemons de:



Img. 4.2.4: Daemons lanzados por robot_preproceso.pl

- **robot_preproceso.pl:** Programa que recorre los directorios locales de microarrays buscando ficheros de nombres (.genesorig), los cuales actualizará añadiendo en un nuevo fichero .genes los símbolos y nombres de cada gen de la microarray.
  Para ello, se recorrerá el directorio donde están los ficheros de microarray: /var/www/cgi-bin/pcop/microarray/
  Para cada directorio, se busca el fichero XX.genesorig correspondiente (XX es el número de la microarray). Una vez encontrado dicho fichero se le pasa al script perl *preproceso.pl* que será el encargado de identificar el tipo de fichero de genes a tratar.
  Los genes del fichero .genesorig pueden estar especificados mediante un código de secuencia, o bien un código UniGene.
  Dependiendo del código origen, se consultará a la tabla unigene o a la tabla sequences.
  Si el fichero origen dispone de genes con código UniGene, bastará con obtener de la tabla unigene el GeneID asociado a dicho código.
  Si por el contrario, el fichero origen trata genes con código de secuencia, habrá que obtener en primera instancia el código UniGene asociado a dicha secuencia de la tabla sequences. Posteriormente, con este código UniGene bastará para obtener el GeneID de la tabla unigene.
- **preproceso.pl:** Programa que a partir de un fichero con códigos de gen, genera uno con los símbolos y nombres asociados a cada gen.
  Inicialmente, se le pasará un fichero y tratará de comprobar el tipo de códigos de gen que porta dicho fichero.
  Si no detecta ningún tipo de código de gen, dejará el fichero tal cual.
  Si detecta códigos de gen del tipo 3' o 5' (códigos de secuencia), obtendrá el símbolo y nombre del gen a partir del código de secuencia, asociado a un UniGene ID, asociado

a su vez a un GeneID del que obtenemos el símbolo y nombre.
Si detecta códigos de gen del tipo Hs.313 (códigos de cluster Unigene), obtendrá el símbolo y nombre del gen a partir del GeneID asociado a dicho código Unigene.

Por último, lanzamos el encargado de actualizar el archivo que lee el applet de java para cargar los nombres y posiciones del *gene network*, con los nombres actualizados de los genes.

- **CopyGenesToBioFactors.pl:** Generaremos un nuevo BioFactors2K5.net, situado en **/var/www/html/applic/gexp/microarray/cruces/perlPrueba/temp/BioFACTORS2K5.net** a partir del archivo actualizado de la microarray de 17.genes, situado en **/var/www/html/applic/gexp/microarray/cruces/perlPrueba/temp/17.genes**.

4.3. Applet de java

El archivo sobre el que tenemos que efectuar los cambio se encuentra en el siguiente path del servidor: **/var/www/html/applic/gexp/microarray/BioProject.jar**. Como podemos observar es un .jar, es decir, un archivo que se puede ejecutar y que contiene todos las clases y .java comprimidos. Para poder hacer cambios sobre BioProject.jar adjunto el anexo1 con un pequeño manual que explica la forma correcta de importar el projecto en Eclipse.

Hemos añadido las funcionalidades que requería el proyecto, como la de permitir hacer consultas a partir de las nomenclaturas de los genes introducidos manualmente, o la de marcar los **genes relacionados** de la aplicación php en el *gene network*, todo esto en el archivo **BioProjectPanel.java**, que es el que tiene las partes del aplicativo que han sido modificadas.

En este applet generamos el nombre del archivo que vamos a enviar a mostrar.php, para que escriba el identificador de los genes en la microarray para marcarlos en *gene network*. El nombre del archivo tiene una estructura como la siguiente:

- **CDGS_1652011181423900:** CDGS_día/mes/año/hora/minuto/segundo/milisegundo

5. Conclusiones, presupuesto y trabajos futuros

5.1. Conclusiones

Tal como se detalla a continuación he conseguido alcanzar todos los objetivos propuestos:

- Objetivos relacionados con el aplicativo php que realiza el cruce de las bases de datos biomédicas para encontrar nuevas relaciones entre los genes:

    - El proceso encargado de los cruces con las bases de datos biomédicas, recibe una serie de **genes** de **entrada**. Ahora el aplicativo permite al usuario introducir manualmente nombres de genes y proteínas, con lo que ahora se pueden buscar relaciones con genes que no aparecen en la *microarray* que se analiza. Si estos genes de entrada han sido introducidos manualmente por el usuario, el proceso de cruces de base de datos se encarga de encontrar el símbolo identificador del gen para todas las diferentes nomenclaturas del gen y de la proteína sintetizada por el gen.

    - De la misma forma, puede mostrar las relaciones génicas partiendo del nombre de una patología, función celular, o un tratamiento. Es decir los genes relacionados esa patología, función celular, o tratamiento. Sin necesidad de unos **genes de entrada**.

    - He desarrollado un algoritmo que minimiza el coste en tiempo de la búsqueda de **relaciones génicas**, optimizando consultas y cruces, y aprovechando las posibilidades que nos ofrece tener una Base de datos local en una capa inferior al aplicativo php.

- Objetivos relacionados con el aplicativo php encargado de operar con las relaciones génicas encontradas:

    - En el proceso de operar con las relaciones **génicas encontradas**, ya no hacemos uso de las herramientas *Eutils*, porque la información la tenemos almacenada en la base de datos local.

    - El proceso de operar con las **relaciones génicas** muestra los **genes relacionados** con los **genes de entrada**. Estos **genes resultado**, pueden estar a su vez relacionados entre ellos. He dotado al aplicativo php encargado de operar con las **relacione génicas**, de la posibilidad de visualizar las **relaciones génicas** que relacionan los **genes resultado** entre ellos además de con los genes de entrada.

    - Se ofrece la posibilidad de visualizar un resumen de todas las **relaciones** que tiene

un **gen relacionado** con respecto a los **genes de entrada**. Esto se consigue aprovechando que la información de las **relaciones génicas** resultante de las consultas está almacenada en tablas auxiliares.

- Actualmente el proceso de operar con las relaciones génicas, tenemos la opción de reordenar los listados de **relaciones génicas** por cualquier base de datos consultada así como por los **genes relacionados** con los **genes de entrada**. De esta forma el usuario puede observar todas las relaciones génicas en las que un gen resultado está involucrado.

- El aplicativo php estructura la información de tal manera que resulta una interfaz muy amigable e intuitiva a los usuarios. Todos los datos están encapsulados en una tabla, Y todas las operaciones sobre los **genes relacionados** y los listados de **relaciones génicas** son ad hoc.

- Objetivos referentes a la base de datos local:

  - En un estudio previo de la Base de datos local he detectado las tablas que tienen más accesos y más volumen de datos, y las he reestructurado dividiendo dichas tablas en varias partes para optimizar los accesos. Esto logra optimizar el proceso de cruce de información que realiza el aplicativo php.

  - He añadido a la base de datos la información que anteriormente se descargaba on-line cuando era necesitada por el proceso de operar con las relaciones génicas del aplicativo php. Esto implica que prescindo del uso de las herramientas *Eutils* en el aplicativo php. Esta información se descarga mediante dichas herramientas *Eutils* pero durante la actualización de la Base de datos local.

  - He añadido la información necesaria a la base de datos para permitir la búsqueda de **relaciones génicas** introduciendo manualmente las posibles nomenclaturas de los genes y las proteínas.

- Objetivos que requería la actualización de la base de datos:

  - La actualización se ejecuta de forma on-line, es decir, en ningún momento se deshabilita la aplicación web por motivos de actualizaciones.

  - La actualización es la encargada de hacer cambios en la estructura de la Base de datos, en caso que esta crezca fuera de unos límites. Es un proceso muy robusto, en el que considero la posibilidad de caída del servidor. Tanto del local, como del remoto.

  - Al final de la actualización, se actualiza el archivo que lee *PCOPGene*[2] para cada

microarray con los nombres oficiales de los genes de esa *microarray*. De esta manera tenemos los nombres de genes siempre actualizados en la *gene network*.

- Objetivos cumplidos en el applet PCOPGene-net :

  - He adaptado el applet de *PCOPGene-net* para permitir al usuario introducir manualmente los nombres de los genes, incluyendo genes que no aparecen en la *microarray* analizada, ya sea mediante el símbolo oficial del gen o diferentes nomenclaturas.

  - El applet está adaptado para marcar en la *gene network* los **genes relacionados** que hemos obtenido en la consulta. Esto se consigue mediante un fichero generado por el proceso de cruce de las bases de datos que es leído por el *PCOPGene*[2].

Como ya he comentado en la sección fases **3.1.2.3.1**, la solución que he implementado para tener un acceso rápido a la base de datos ha sido dividir las tablas más grandes y con más accesos. No he dividido el resto de tablas ya que no tienen suficiente volumen de información, y dividir estas tablas supondría preparar el acceso a estas tablas mediante bucles anidados. El acceso mediante bucles anidados, supone un coste de tiempo más grande del que actualmente se utiliza para acceder a las tablas (que es el acceder mediante consultas simples a la base de datos directamente a estas tablas que no están divididas) y no resulta óptimo. De cara a un futuro, hay que mantener un estudio del crecimiento de estas tablas y tomar la decisión de implantar un sistema de tablas divididas si fuera necesario.

Hasta el momento en que encontré la solución definitiva a los diferentes problemas (como la división de las tablas con mas transito de información), he implementado diferentes alternativas pero que resultaron contraproducentes. Estas alternativas aparecen explicadas en la sección fases. Las limitaciones y paradojas encontradas en el desarrollo de estas alternativas, me han servido para dar con la solución finalmente óptima.

Me gustaría resaltar que la fase que más tiempo me ha llevado solucionar y más dificultad ha conllevado ha sido el diseño del cruce on-line de las diferentes bases de datos. Esta dificultad radica en que ha sido necesario tanto el estudio a fondo de la aplicación web como de la estructura de la base de datos, además de la prueba de numerosas alternativas, que aunque eran atractivas en un principio, por un motivo u otro acababan siendo contraproducentes a la práctica.

He empleado muchas horas en este trabajo para solucionar todos los problemas que han ido surgiendo e intentar predecir los posibles problemas que podían surgir. Sin embargo, al ir encontrando solución a todos estos problemas que surgían, ha resultado un proyecto aunque

agotador por un lado, muy motivador por el otro. Aparte de esto, el diseño de bases de datos es un campo que me interesa mucho de cara a mi vida profesional y gracias a este proyecto he adquirido una buena experiencia ya que es el primer caso real, complejo y profesional, con el que me he encontrado.

5.2. Presupuesto

Presupuesto del proyecto			
Fases	**Descripción**	**Horas empleadas**	**Coste (€)**
Fase 1: *Estudio del problema e implementación de la solución en el aplicativo web*	Conocimientos	100	800
	Estudio PCOPGene	10	80
	Estructura del aplicativo web actual	10	80
	Estudio de la base de datos local	10	80
	Pruebas de consultas contra la base de datos local	10	80
	Implementación de posibles soluciones	100	800
	Reorganización de la base de datos local	60	480
	Implementación de la solución pactada	80	640
	Test de pruebas del aplicativo web	35	280
Fase 2: *Creación y programación de la actualización trimestral*	Programación de los daemons que actualizan la base de datos	20	160
	Test de la actualización total	40	320
Fase 3: *Modificación del applet java*	Modificación del applet java	20	160
	Test de pruebas del aplicativo final	24	192
Fase 4:	Informe técnico	15	120

Documentación			
Herramientas utilizadas		**Coste (€)**	
Servidor Web Apache		Open Source	
Motor gestor de bases de datos MySQL		Open Source	
Eclipse (programación Java)		Open Source	
Interpretes de Perl (descarga de modulos necesarios de Perl)		Open Source	
Sistema operativo Unix		Open Source	
Precio total del proyecto (534h totales 8€/h).....4272 €			

Coste del servidor utilizado	
Serv. Supermicro SYS-6016T-NTRF • RAID 10 con 4 HDD de 300GB • RAID 5 con 4 HDD de 5.2 TB • SuperServer SYS-6016T-NTRF • Intel DPWestmere 6C X5650 • DDR3 133 8 GB • HUA722020ALA330 Hitachi 2 TB SATA II • Adaptec RAID 5405 SATA/SAS Kit PCI-E	1 2 8 4 1
Precio del servidor utilizado en el proyecto.....8000,29 €	

*El precio del servidor, es el que he usado para desarrollar el proyecto. Pero el proyecto puede implantarse en sistemas menos potentes.

5.3. Trabajo futuro

Mantener un estudio del crecimiento de la Base de datos local, por si en un futuro crecen excesivamente las tablas que no han sido divididas en este proyecto por cuestiones de eficiencia.

Mantener un estudio de las bases de datos remotas por si se puede ampliar la base de datos local con información nueva que aumente las posibilidades de encontrar más relaciones génicas.

Se tendría que habilitar el estudio de diferentes *microarrays* simultaneamente, de esta forma podríamos establecer relaciones génicas entre genes de una *microarray* y otra, aunque estas *microarrays* estudiasen experimentos distintos. Para ello habría que modificar *PCOPGene[2]*, y encontrar el modo de cargar varias *microarrays* así como realizar los cruces entre los genes de estas.

Actualmente la aplicación de *PCOPGene[2]* sólo permite marcar los **genes relacionados** de la última consulta de relaciones génicas que ha ejecutado el usuario. Sin embargo habría que habilitar un histórico de las consultas que ha hecho cada usuario para cada *microarray*, y permitir al usuario recuperar los resultados de una consulta anterior, tanto para visualizarla como para que se marquen los **genes relacionados** en la *gene network.* Permitir que el usuario vaya cambiando de archivo, a la par que va haciendo nuevas consultas.

Otra opción sería poder escoger en el proceso de operar con las **relaciones** génicas del aplicativo php, aquellos **genes** relacionados que desea marcar el usuario en el *gene network*. De esta forma, sólo marcaría en el gene netwo*rk* los genes de real interés por parte del investigador.

6. Anexos

6.1. Anexo1: Manual de importación de proyecto a Eclipse

1. File-new-project, seleccionar java project , escribir el nombre del project y en el apartado de project layout seleccionar la primera opción, "**Use project folder as root for source and class files** ".

2. Sobre el project creado, botón derecho **buildpath-newsourcefolder** y de nombre "src".

3. Sobre la carpeta creada en el 2, "src". botón derecho import-archivefile, cargar el bioproject.jar. desmarcar la opción de "**Overwrite existing resource without warning**".

4. Si el build está automático, saldrán directamente los errores sino "**build all**".

5. Cargar librerías 1 a 1, las tenemos en el servidor para descargar, sobre el project, botón derecho buildpath-addexternalarchives y se cargan las librería sucesivamente para la **colt, common, jung, log** y **mysql**.

6. Hacer el paso 4 y en lugar de errores aparecen warnings.

7. Modificas lo necesario.

8. Guardar y hacer el paso 4.

9. sobre el project, botón derecho export-jarfile, dejamos marcados los archivos classpath y project, marcar del primer bloque, la primera opción "**Export generate class files and resources**" y la tercera "**Expor java source files and resources**". Del bloque de abajo marcar la primera "**Compress the contentes of the JAR file**" y la tercera "**Overwrite existing files without warnings**" poner el nombre del jar y finish.

   El archivo jar generado ya lo se puede subir al servidor con los cambios realizados.

7. Bibliografía

[1] Instituto de Biotecnología y de Biomedicina (IBB) de la Universidad Autónoma de Barcelona. http://ibb.uab.es/ibb /

[2] Huerta M, Cedano J, Peña D, Rodriguez A, Querol E. (2009) PCOPGene-Net: holistic characterisation of cellular states from microarray data base on continuous and non-continuos analysis og gene-expression relationships BMC Bioinformatics 2009 May

[3] **Cedano J, Huerta M, Querol E. (2008)** NCR-PCOPGene: An Exploratory Tool for Analysis of Sample-Classes Effect on Gene-Expression Relationships Advances in Bioinformatics, vol. 2008

[4] D**elicado, P.(2001)** Another look at principal curves and surfaces. Journal of Multivariate Analysis, 77, 84-116

[5] **Delicado, P. and Huerta, M. (2003**): 'Principal Curves of Oriented Points: Theoretical and computational improvements'. Computational Statistics 18, 293-315.

[6] **Cedano J, Huerta M, Estrada I, Ballllosera F, Conchillo O, Delicado P, Querol E.** (2007) A **web** server for automatic analysis and extraction of relevant biological knowledge. Comput Biol Med. 37:1672-1675.

[7] http://revolutionresearch.uab.es : Web server for on line microarray analysis supported by the Institute of Biotechnology and Biomedicine of the Autonomous University of Barcelona (IBB-UAB).

[8] **Huerta M, Cedano J, Querol E. (2008)** *Analysis of nonlinear relations between expression profiles by the principal curves of oriented-points approach.* J Bioinform Comput Biol. 6:367-386.

[9] La Web oficial del National Center for Biotechnology Information (NCBI) http://www.ncbi.nlm.nih.gov/

[10] KEGG (Kyoto Encyclopedia of Genes and Genomes) http://www.genome.jp/

[11] Buscador web mediante el que accedemos a las bases de datos del NCBI.
http://www.ncbi.nlm.nih.gov/gquery/gquery.fcgi

[12] Herramientas WebService gratuitas del NCBI
http://eutils.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html

[13] National Center for Biotechnology Information (US); 2010. Entrez Programming Utilities Help Bethesda (MD)

[14] La Web oficial de uso de SQL
http://www.sql.org/

[15] Motor gestor de bases de datos OpenSource.
http://www.mysql.com/

[16] Web oficial con la API de php
http://php.net/

[17] Diseño CSS de la tabla: Estilos CSS para el Diseño Web. Universitat Pompeu Fabra, prof. Alejandro Ramírez.

[18] Web oficial con documentación de Perl
www.perl.com

[19] Web oficial de descarga de todos los módulos existentes de Perl.
http://search.cpan.org/

[20] Web oficial de Eclipse
http://www.eclipse.org/

Resumen

El IBB ha desarrollado un servidor de aplicaciones: http://revolutionresearch.uab.es para el análisis de microarrays. En este servidor podemos encontrar una aplicación web, *PCOPGene,* que permite estudiar y analizar *microarrays* con diversos métodos desarrollados en el IBB.

La tecnología de *microarrays,* nos proporciona matrices que contienen filas que representan los genes y columnas que representan los experimentos. En cada celda tenemos el nivel de expresión de dicho gen para dicho experimento.

El presente proyecto trata entonces de como optimizar y ampliar significativamente las funcionalidades de distintas herrameintas de *PCOPGene*, para poder relacionar genes más allá de las relaciones de expresión de la microarray que se está analizando. Esto se consigue usando información biomédica de todo tipo. Esta información está inicialmente almacenada en diferentes bases de datos remotas.

Resum

El IBB ha desenvolupat un servidor de aplicacions: **http://revolutionresearch.uab.es** per l'anàlisis de microarrays. En aquest servidor podem trobar una aplicació web, *PCOPGene*[2], que permet estudiar y analitzar microarrays amb diversos mètodes desenvolupats al IBB.

La tecnología de microarrays, ens proporciona matrius que contenen files que representen els gens i columnes que representen els experiments. En cada cel·la tenim el nivell de expresió de cada gen amb cada experiment.

En el present projecte, tracta de com optimitzar i ampliar significativament les funcionalitats de diverses eines de *PCOPGene*[2], per poder relacionar gens més enllà de les relacions de expressió de la microarray que s'està analitzant. Això es consegueix utilitzant informació biomédica de tot tipus. Aquesta informació està inicialment emmagatzemada en diferents bases de dades remotes.

Summary

The IBB center, has developed a application server: http://revolutionresearch.uab.es to analyze the microarrays. In this application server we can find a web application,

PCOPGene[2],this application allows to study and analyze microarrays with many methods developed in the IBB.

The microarrays technology provide us a matrix that contain rows witch represent genes, and columns that are the experiments. In each cells, we obtain the expression of the gene for each experiments.

This project aims to significantly expand the capabilities of PCOPGene [2], to find the gene relations beyond expressions relations on the microarray. We obtain the information to fin the genes relations in a remote biomedical databases.